ELEC 490/498 Final Report

Group 50

Sustainable Supercomputing: Using power controls to maximize performance and minimize energy usage

James Corley 20166272

Simon Dudtschak 20166103

Alex Perrin 20056483

To: Dr. Ryan Grant, Dr. Alireza Bakhshai,

Dr. Mike Korenberg, Dr. Alex Tait, Dr. Sean Whitehall

Submitted: April 10th 2022

Executive Summary

High-performance computing (HPC) is a field in which almost all other scientific domains rely. It is thus unsurprising with this large demand that these clusters run almost constantly, consuming a massive amount of energy in doing so. As such, any improvement to energy usage or job runtime has the potential to have a broad impact, both in time efficiency and sustainability. This project's goal is to investigate the potential improvements to these two metrics through the implementation of region hint behaviour. These hints identify sections of code as computationally intensive in different ways and use dynamic frequency scaling to tune Central Processing Unit (CPU) core speeds to make the most efficient use of the hardware according to the type of computation being executed. For this project, five hint behaviours were created, including serial, parallel, memory-bound, input/output (I/O), and network communication. For each of these cases, testing was completed on a reserved node on the Queen's Center for Advanced Computing's (CAC) Frontenac cluster. The implementation of region hint behaviour in these tests saw an average of 6.5% faster runtimes and a 9.4% reduction in energy use. Moreover, three general conclusions to improve runtimes and energy usage can be drawn from this project. Firstly, any program with multiple available cores that is executing a serial portion of code should maximize the clock speed of one core and limit all others. Secondly, any time the frequency is likely to operate within a specific range the minimum and maximum frequency bounds should be tightened to that range. Thirdly, if a program's runtime is not CPU-bound, its core frequencies should be decreased to a degree. These findings were compiled in the hopes of contributing to the Power API, a community led tool for power monitoring and control. Specifically, the hint behaviours within this report should be implemented in the Power API's region hint framework to allow for more time and energy efficient scientific computing.

Table of Contents

1. Introduction

As the computational needs of the world rise, the energy efficiency of computers will need to rise as well. While advancements in hardware will certainly play a significant role, it is important that software solutions exist to fully utilize hardware, both for current and future technologies. By optimizing the performance of hardware with standardized hardware-agnostic software, high-performance computing systems can reinvest energy savings into more computation and improved systems.

1.1 Background

The basis of this project is the Power API [1], a community-lead API for monitoring and controlling energy usage in computing systems. Power API as it exists today outlines a number of Role/System pair interfaces, which suggest attributes and functions that should be supported by an implementation. Most important to this project is the “Application, Operating System” interface, which describes core functions to control the power and frequency limits, the high-level functions surrounding hint regions, and even provides function prototypes for future implementation. Hint regions are intended to indicate that the code of an application using the Power API is entering a region with a specific style of code (Serial, Parallel, Compute, Communicate, I/O, and Memory Bound); these regions are described in detail in 3.2 Hint Implementation. The implementation of communicating these hint regions to the operating system is left to the implementer.

This project aims to maintain Power API’s architecture-agnosticism, as discussed in section 7.3 of the Power API documentation [1]. This is to say that the goal of this project is to have it be applicable to a system regardless of the type of CPU being utilized. Hardware by different manufacturers often have similar capabilities but with different implementations, such as `intel_pstate` versus `amd_pstate`; by accommodating all, or as many architectures as possible, the API can be made more portable, which reduces the barrier to wider adoption. While the tests done throughout the project are biased towards Intel architecture, the findings still apply broadly and can be used in the development of a non-architecture dependent tool.

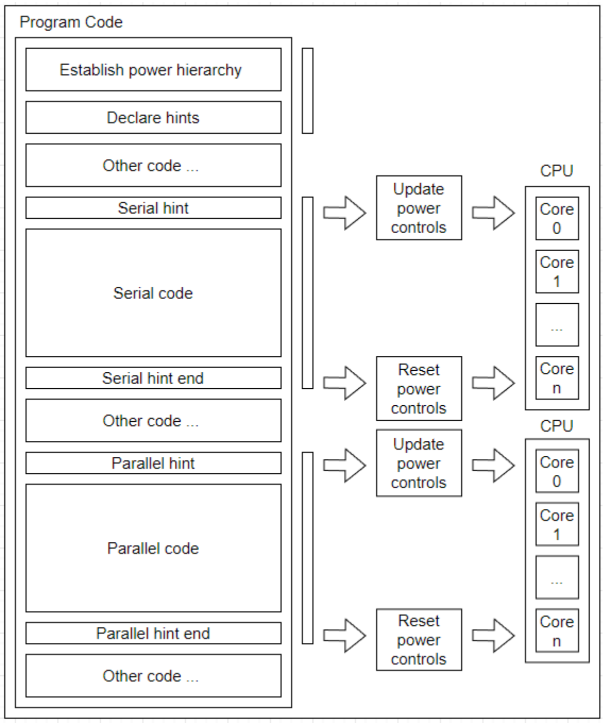


Figure 1: Example of region hint behaviour in a program

1.2 Motivation

This project is motivated to improve the efficiency of high-performance computing systems, through the use of CPU power controls. As the original documentation for the Power API expresses, there is an existing demand for software-based tuning of high-performance computing systems. A detailed example is the *Power/Energy Use cases for High Performance Computing* paper [2], which provides several use cases and specifications. Several broadly applicable scenarios are described in the appendix of the document, though the scenario most similar to the workings of this project is “B.2 Increase Application Efficiency”. By measuring baseline energy and power metrics, then finely tuning parameters, a repeated application can be made to run as efficiently as possible.

As described in 1.1 Background, the hint region interface currently exists only as a suggestion of standardization. As-is, the API does not communicate the hints to the operating system or power governors in any way. The aim of this project then is to research and implement a set of behaviors tied to the default hint cases. The primary goal is the research and refinement of optimal behaviors that increase efficiency in the existing hint regions.

The ultimate motivation of this project is positive environmental impact made possible by providing knowledge and user-friendly tools to improve the efficiency of large-scale HPC systems. HPC systems are capable of consuming huge amounts of electricity; by reducing the energy required those savings can be passed on as less power generated by fossil fuels, and eventually as positive environmental impact.

2. Design

2.1 Functional Requirements

The first functional requirement is that the solution must be able to control necessary operational characteristics of the CPU that may affect energy consumption. This includes CPU core frequency min/max, frequency scaling ratios, p-state, and c-state, power limit. The solution must be able to enforce CPU behavior in way that is reliable and reproducible and to the degree that it is able, must not conflict with other systems that may enforce CPU behavior in unintended ways.

The solution is required to utilize control over CPU behavior to optimize for regions of code defined as serial, parallel, memory bound, I/O, and network.

The solution must also measure CPU power consumption and runtime to evaluate the performance of the hint behavior. The solution must minimize instrumentation overhead in these measurements, as well as take measurements at a degree of precision necessary for evaluation.

The solution must be compatible with HPC environments and so must support Intel and AMD devices. Must be compatible with operating systems and software common to HPC systems such as Linux.

2.2 Constraints

Permissions

The primary constraint for this project is the issue of permissions. In order to manually interact with these power controls, it requires either accessing and changing protected interfaces, or using programs to do so. On personal machines, these system protections can mostly be bypassed by changing file permissions, using driver tools, and by using the `sudo` command. Unfortunately, local machines are not the intended deployment environment for this solution. The actual deployment environment, scientific computing clusters, do not allow these same bypasses to protect their systems. Instead, a special reserved node or group of nodes is required where expanded permissions can be granted. So, while there are solutions to these issues, it ultimately constrains the development process and limits the breadth of testing that can be completed.

Limits of control

The core of this project is the manual control of CPU frequency scaling, however, this control is not absolute. Instruction can be given in the user space but ultimately the CPU often has the final say on its own behaviour. This can cause situations where, unknown to the user, it will not behave as instructed, which can lead to inconsistent benchmarking and other confusing results. While this hidden behaviour is useful in keeping the CPU safe, it can cause complications and must be considered.

Benchmarking

All the findings from this project are based on the collection of data through running several benchmark files which are intended to stress different elements of the computer. However, while these files were designed to be able to stress certain elements, it is difficult to ensure to what degree they are actually doing so. The group still has confidence in its findings and conclusions, but this is an element that was considered.

2.3 Design Process

The design process for this project was complex, since there were many challenges to overcome before it the solution could be started. As such the process will be broken into three main stages of development, discussed below.

The first stage of this design process involved setting up the environments and drivers to begin work on the project. Initially Windows Subsystem for Linux (WSL) was used as the development environment. Unfortunately, windows blocks interaction with the necessary interfaces and counters, making it necessary to have a complete Ubuntu install instead. More than just the environment, the group needed to set up a functional frequency scaling driver and install and setup the Power API. Since this task overlapped with the realization the WSL would not work, this process was quite lengthy, and caused the removal of integration with the Power API from the scope of the project. The final aspect of this stage was the testing and research to begin to read power information and successfully set CPU core frequencies.

The second stage involved initial testing and data collection on personal computers. At this time the development process is that shown in Figure 2. This was used for the development of the serial and parallel cases and was done with `ACPI_cpufreq` and the `CPUpower` toolset.

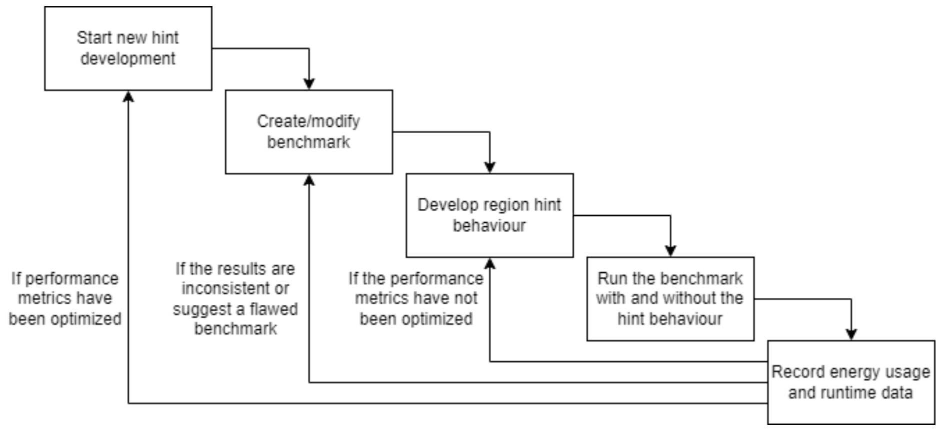

Start new hint development
Create/modify benchmark
Develop region hint behaviour
Run the benchmark with and without the hint behaviour
Record energy usage and runtime data
If performance metrics have been optimized
If the results are inconsistent or suggest a flawed benchmark
If the performance metrics have not been optimized

Figure 2: Hint development workflow

Once access to the node on the CAC Frontenac cluster was granted, the third and final stage began. During this period the design process was the same as listed above, but it was done on the CAC node and with the `intel_pstate` driver active. This is where the design process became optimized, and the final results were gathered.

3. Implementation

3.1 Frequency Scaling

sysfs is a pseudo file system provided by the Linux kernel that exports information about various kernel subsystems, hardware devices, and associated device drivers from the kernel's device model to user space through virtual files. [6]

The region hints are called in line with the program, which exists within the user space. Control over the CPUs behavior such as power and frequency are handled within the kernel space. Through sysfs we can interface with kernel devices such as the CPU frequency scaling driver. The relevant interfaces are described below:

```
/sys/devices/system/cpu/cpu*/cpufreq/
├── scaling_driver
├── scaling_available_governors
├── scaling_governor
├── scaling_setspeed
├── scaling_max_freq
├── scaling_min_freq
└── scaling_cur_freq
/sys/devices/system/cpu/intel_pstate/
└─ status
```

CPU Scaling Drivers

This system is the part of the operating system which enforces CPU behaviour such as clock speed. There exist multiple scaling drivers but only one specific is in use on any given system and can be identified using the `scaling_driver` `sysfs` interface. The scaling drivers that were tested will be discussed below.

intel_pstate is the default scaling driver for modern CPUs and so is important that our solution implements it. Modern intel CPUs regulate performance using P-States which are predefined CPU frequency ranges. Even if the scaling driver selects a single P-State, the actual frequency the processor will run at is selected by the processor itself [6]. When using the `intel_pstate` scaling driver we can control CPU frequency through the `scaling_max_freq` and `scaling_min_freq` interfaces. This means the frequency cannot be set exactly, but the minimums and maximums can be set in a way to force the frequency to a certain range. This allows for the same degree of control since that range can be very small, functionally setting it to a single value.

CPUFreq is the other driver that our solution implements because it is default in most other common configurations. It can come in the form of `acpi_cpufreq` and `intel_cpufreq`. (`intel_pstate` scaling driver selects "passive mode" aka intel_cpufreq for CPUs that do not support hardware-managed P-states (HWP), i.e. Intel Core i5th generation or older) [7] When using the CPUfreq driver we can use the `scaling_setspeed` interface to control CPU frequency. Unlike `intel_pstate`, we can set a specific clock speed for the CPU cores to operate at.

Scaling Governors

In a default configuration the CPU scaling driver employs a `scaling_governor`. These governors are different profiles that enforce the CPU frequency. Table 1 lists some of these governors in the drivers used for this project.

Table 1: acpi_cpufreq and intel_pstate governors

`acpi_cpufreq` Governors	`intel_pstate` Governors
• `performance` • `powersave` • `userspace`	• `performance` • `powersave`

• `ondemand` • `conservative` • `shedutil`	

`userspace` allows for the manual control of CPU frequency through the sysfs interfaces `scaling_setspeed` or `scaling_max_freq/scaling_min_freq`. In other governor modes, the scaling driver will not consider parameters set within sysfs.

CPUpower

CPUpower is a toolset used for interacting with scaling drivers. It has many utilities, including the manipulation of core frequencies with frequency-set, and the changing of governors with governor-set among other functions. This toolset uses high level commands, so no interface needs to be interacted with directly. To set core speeds, `frequency-set` with the `-c` option to define which cores are being changed was used. A diagram describing CPUpower dataflow is given in Figure 3.

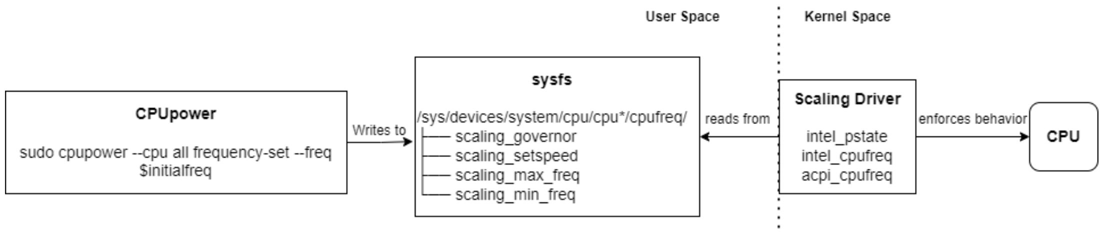
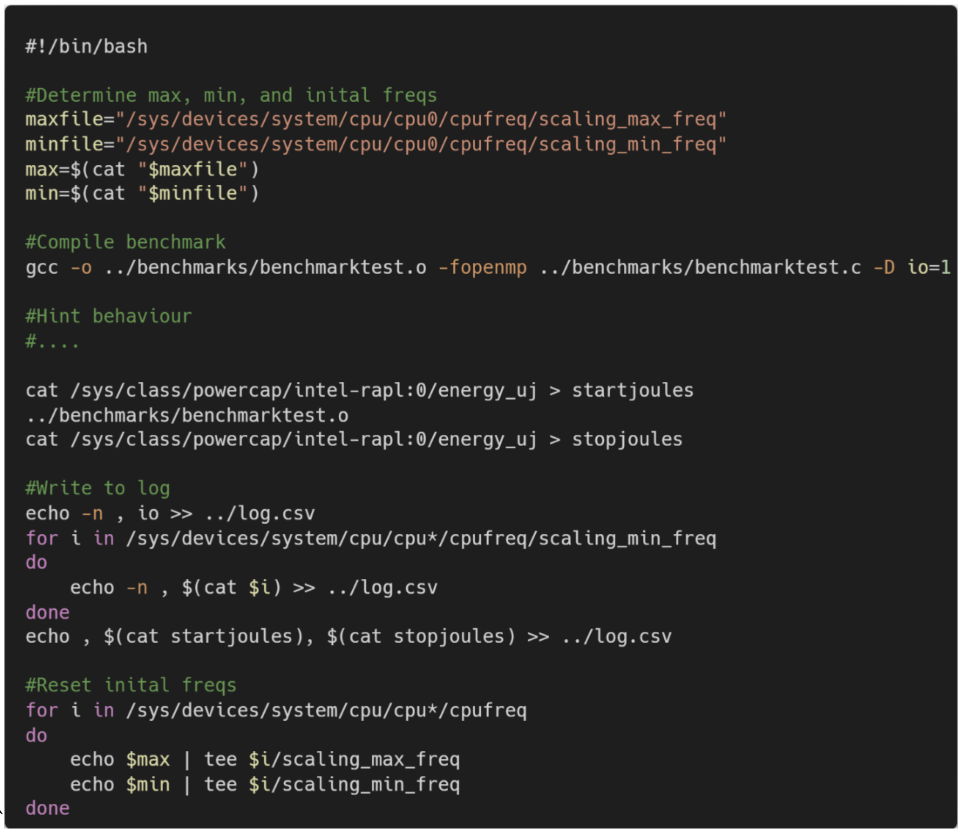


Figure 3: CPUpower dataflow diagram

3.2 Hint Implementation

The solution for this project was implemented in batch files which would contain commands to interact with power controls, run benchmarks, and collect data. The general outline of which is shown in Figure 4. The section with the comment `#Hint behavior` is where the particular hint behavior was implemented and is discussed on a per hint basis in the following sections.

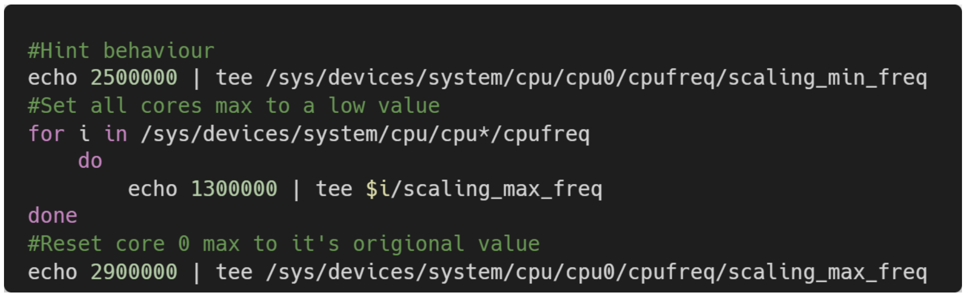


```bash
#!/bin/bash

#Determine max, min, and inital freqs
maxfile="/sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq"
minfile="/sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq"
max=$(cat "$maxfile")
min=$(cat "$minfile")

#Compile benchmark
gcc -o ../benchmarks/benchmarktest.o -fopenmp ../benchmarks/benchmarktest.c -D io=1

#Hint behaviour
#....

cat /sys/class/powercap/intel-rapl:0/energy_uj > startjoules
../benchmarks/benchmarktest.o
cat /sys/class/powercap/intel-rapl:0/energy_uj > stopjoules

#Write to log
echo -n , io >> ../log.csv
for i in /sys/devices/system/cpu/cpu*/cpufreq/scaling_min_freq
do
    echo -n , $(cat $i) >> ../log.csv
done
echo , $(cat startjoules), $(cat stopjoules) >> ../log.csv

#Reset inital freqs
for i in /sys/devices/system/cpu/cpu*/cpufreq
do
    echo $max | tee $i/scaling_max_freq
    echo $min | tee $i/scaling_min_freq
done
```

Figure 4: Hint testing batch file

Serial

The serial regions of code are those containing high volumes of code that must be executed in serial, such as loops or large non-parallelizable computation. As such the behaviour for these regions is to minimize all cores frequency down to their minimum, and then push one core, the core running the computation, to it's maximum. This reduces power consumption on the non-active cores, and increases efficiency of the active core, since it will not be thermally limited by other cores producing heat, and as such can go into a turbo state.

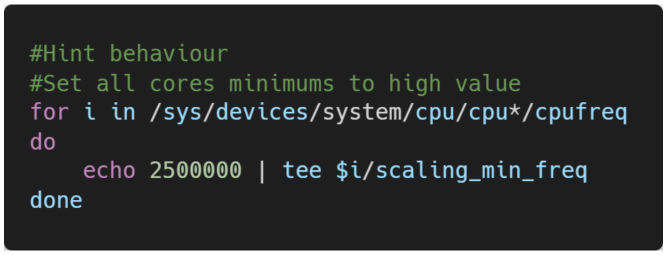


```
#Hint behaviour
echo 2500000 | tee /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
#Set all cores max to a low value
for i in /sys/devices/system/cpu/cpu*/cpufreq
    do
        echo 1300000 | tee $i/scaling_max_freq
done
#Reset core 0 max to it's origional value
echo 2900000 | tee /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

Figure 5: Serial hint behaviour

Parallel

Parallel regions will be a common case in HPC settings, and simply consists of regions where the computation is being split into multiple threads and executed across several cores. In these areas, since all the cores will be executing code, none of their performance should be limited. Instead, their minimum clock frequency can be increased to ensure they are all performing at a high speed constantly.



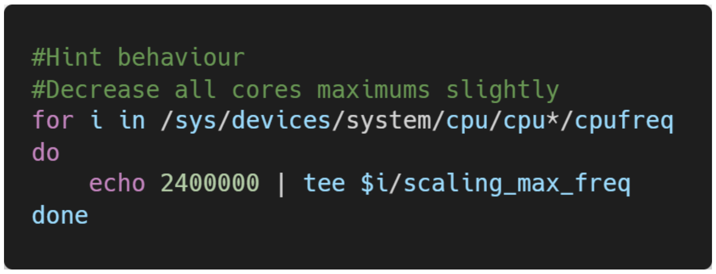
```
#Hint behaviour
#Set all cores minimums to high value
for i in /sys/devices/system/cpu/cpu*/cpufreq
do
    echo 2500000 | tee $i/scaling_min_freq
done
```

Figure 6: Parallel hint behaviour

Memory bound

The memory bound regions of a program are those where the time for that region to execute is primarily dependent upon the memory usage instead of the number of operations in that region. As a result, to improve efficiency, the CPU frequencies can be lowered to a point just faster than it would become CPU bound. If this is done properly it will not impact runtime, since it will still be memory bound, but it will have a slight reduction in power consumption because the cores will be operating at a frequency closer to the boundary between CPU and memory bound.

Depending on how memory versus CPU intensive a region is, the optimal frequency to reduce to will change, so finding an optimal reduction across all cases would be quite challenging. In this project, an upper limit was found for the particular benchmark used, but this is likely not perfect for all situations.

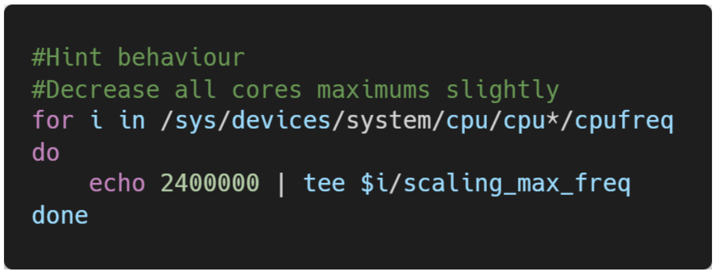


```
#Hint behaviour
#Decrease all cores maximums slightly
for i in /sys/devices/system/cpu/cpu*/cpufreq
do
    echo 2400000 | tee $i/scaling_max_freq
done
```

Figure 7: Memory bound hint behaviour

Input/Output

The I/O regions of code are similar to those memory bound regions in that their execution may be limited based on external devices having slower speeds than the CPU. As a result, it was expected that the approach to the solution would be similar. Interestingly, the most optimal results were achieved with the exact same settings as the memory bound case. It was expected that a larger overall decrease to core speeds would be better, because interacting with RAM is faster than an I/O device like a hard disk drive but this was not the case.

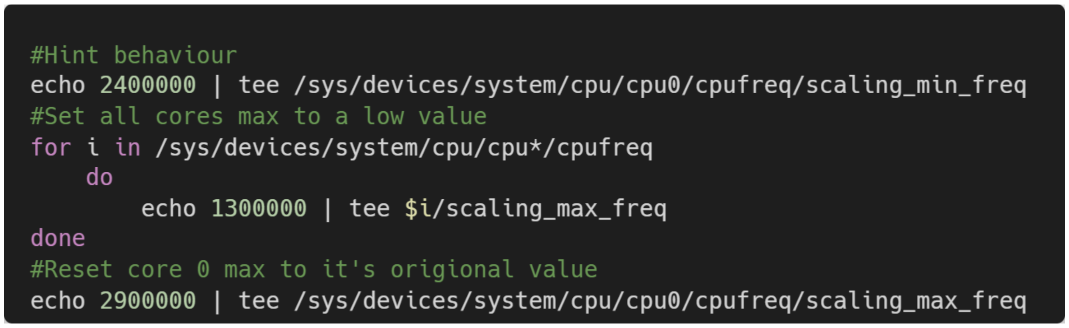


```
#Hint behaviour
#Decrease all cores maximums slightly
for i in /sys/devices/system/cpu/cpu*/cpufreq
do
    echo 2400000 | tee $i/scaling_max_freq
done
```

Figure 8: I/O hint behaviour

Network communication

Several different behaviours were tried for this case, but the configuration that yielded the best results was almost the exact same as the serial case. In this situation, the primary constraint should be the network, which would cause the CPU to need to wait for data to be sent or received. However, through testing it was apparent that the CPU was highly active at specific times, and so during those times it required a high frequency to prevent delay. The best solution was very similar to the serial case, and consisted of boosting one core's minimum clock frequency, and decreasing all the other core's maximums. Notably, this does operate under the assumption that most network accesses will not be done in parallel operations. This is assumed for two reasons, firstly, parallel sections should use the parallel hint instead of this one, and second, in the majority of scenarios they will be done large network accesses can be done in serial regardless. For example, openMPI can be used for the distributed memory parallelization, where this hint may be helpful, and then openMP can be used for shared memory parallelization where the parallel hint can be used.



```
#Hint behaviour
echo 2400000 | tee /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
#Set all cores max to a low value
for i in /sys/devices/system/cpu/cpu*/cpufreq
    do
        echo 1300000 | tee $i/scaling_max_freq
done
#Reset core 0 max to it's origional value
echo 2900000 | tee /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

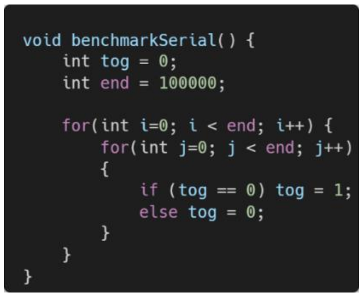
Figure 9: Network communication hint behaviour

4. Testing

4.1 Benchmarks

Serial Benchmark

The serial benchmark is simple and only contains two nested for loops which update a variable for a given number of iterations. While it does not contain complex computation, it still illustrates an environment where there is a lengthy serial task that only requires a single core to execute.

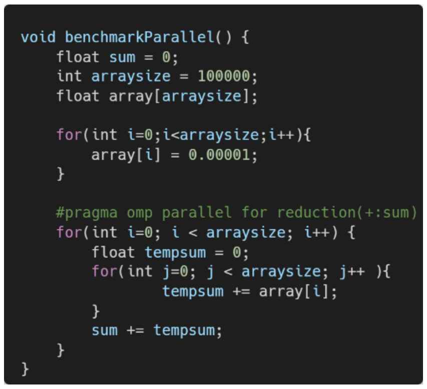


```
void benchmarkSerial() {
    int tog = 0;
    int end = 100000;

    for(int i=0; i < end; i++) {
        for(int j=0; j < end; j++)
        {
            if (tog == 0) tog = 1;
            else tog = 0;
        }
    }
}
```

Figure 10: Serial hint benchmark

Parallel Benchmark

The parallel benchmark uses openMP to parallelize a summation for loop and then executes a reduction across it to sum the individual threads values. While this is another relatively simple benchmark, it will still manage to emulate a task parallelized for shared memory environment.

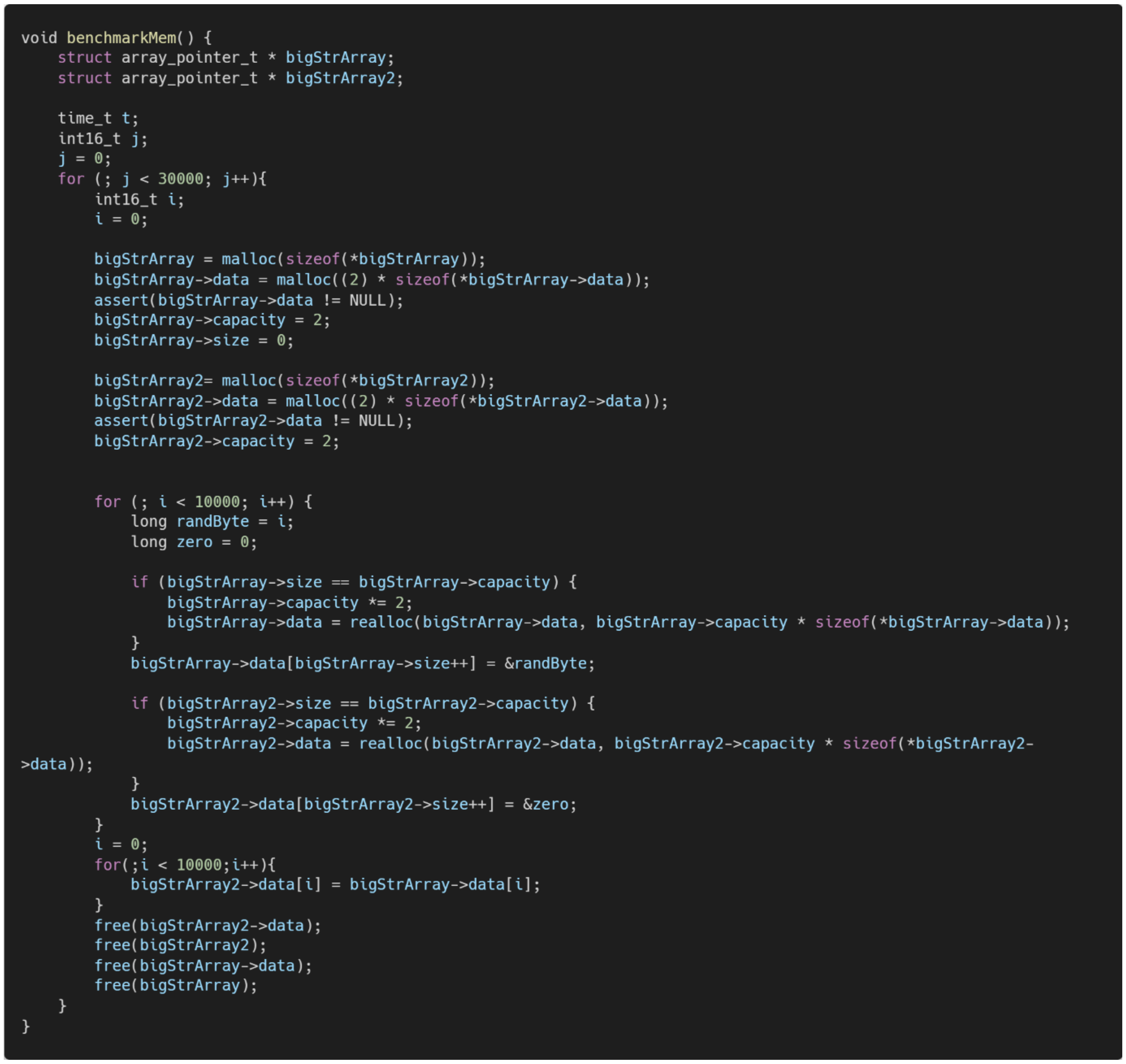


```
void benchmarkParallel() {
    float sum = 0;
    int arraysize = 100000;
    float array[arraysize];

    for(int i=0;i<arraysize;i++){
        array[i] = 0.00001;
    }

    #pragma omp parallel for reduction(+:sum)
    for(int i=0; i < arraysize; i++) {
        float tempsum = 0;
        for(int j=0; j < arraysize; j++ ){
                tempsum += array[i];
        }
        sum += tempsum;
    }
}
```

Figure 11: Parallel hint benchmark

Memory-bound Benchmark

The memory bound benchmark used creating and writing to arrays as it's method of stressing memory. The benchmark consists of declaring two arrays, and then iterating through them, and adding new elements, each time reallocating memory to create a larger array to fit the new value. Then, once they are filled, it iterates through them again and copies the information from one into the other. This high volume of memory operations should put a large load on memory and make the runtime of the program defined largely by the memory speed.

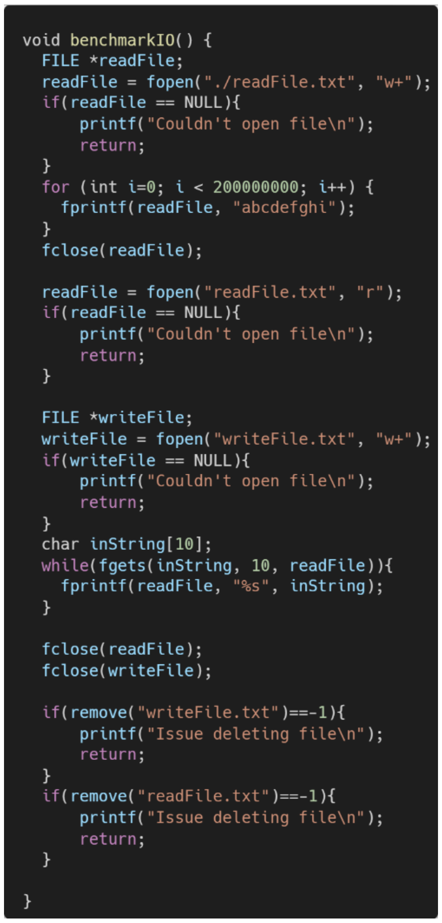


```c
void benchmarkMem() {
    struct array_pointer_t * bigStrArray;
    struct array_pointer_t * bigStrArray2;

    time_t t;
    int16_t j;
    j = 0;
    for (; j < 30000; j++){
        int16_t i;
        i = 0;

        bigStrArray = malloc(sizeof(*bigStrArray));
        bigStrArray->data = malloc((2) * sizeof(*bigStrArray->data));
        assert(bigStrArray->data != NULL);
        bigStrArray->capacity = 2;
        bigStrArray->size = 0;

        bigStrArray2= malloc(sizeof(*bigStrArray2));
        bigStrArray2->data = malloc((2) * sizeof(*bigStrArray2->data));
        assert(bigStrArray2->data != NULL);
        bigStrArray2->capacity = 2;


        for (; i < 10000; i++) {
            long randByte = i;
            long zero = 0;

            if (bigStrArray->size == bigStrArray->capacity) {
                bigStrArray->capacity *= 2;
                bigStrArray->data = realloc(bigStrArray->data, bigStrArray->capacity * sizeof(*bigStrArray->data));
            }
            bigStrArray->data[bigStrArray->size++] = &randByte;

            if (bigStrArray2->size == bigStrArray2->capacity) {
                bigStrArray2->capacity *= 2;
                bigStrArray2->data = realloc(bigStrArray2->data, bigStrArray2->capacity * sizeof(*bigStrArray2-
>data));
            }
            bigStrArray2->data[bigStrArray2->size++] = &zero;
        }
        i = 0;
        for(;i < 10000;i++){
            bigStrArray2->data[i] = bigStrArray->data[i];
        }
        free(bigStrArray2->data);
        free(bigStrArray2);
        free(bigStrArray->data);
        free(bigStrArray);
    }
}
```

Figure 12: Memory bound benchmark

I/O Benchmark

To represent input and output operations, the I/O benchmark used file reading and writing to external memory. The read file was created first, and had a large string written to it before closing it and opening the write file. Then the read file was opened again, and the entire string was read out from it, and written into the write file.

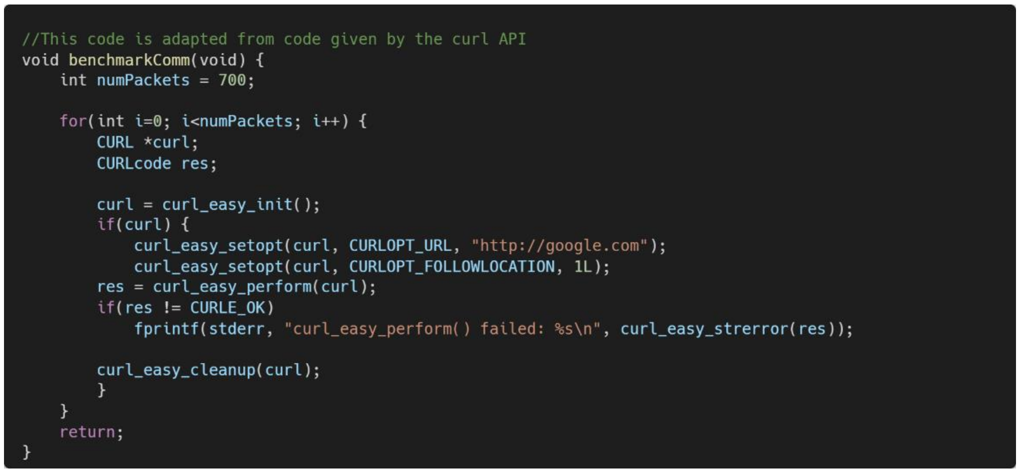


```c
void benchmarkIO() {
  FILE *readFile;
  readFile = fopen("./readFile.txt", "w+");
  if(readFile == NULL){
      printf("Couldn't open file\n");
      return;
  }
  for (int i=0; i < 200000000; i++) {
    fprintf(readFile, "abcdefghi");
  }
  fclose(readFile);

  readFile = fopen("readFile.txt", "r");
  if(readFile == NULL){
      printf("Couldn't open file\n");
      return;
  }

  FILE *writeFile;
  writeFile = fopen("writeFile.txt", "w+");
  if(writeFile == NULL){
      printf("Couldn't open file\n");
      return;
  }
  char inString[10];
  while(fgets(inString, 10, readFile)){
    fprintf(readFile, "%s", inString);
  }

  fclose(readFile);
  fclose(writeFile);

  if(remove("writeFile.txt")==-1){
      printf("Issue deleting file\n");
      return;
  }
  if(remove("readFile.txt")==-1){
      printf("Issue deleting file\n");
      return;
  }

}
```

Figure 13: I/O hint benchmark

Network Communication Benchmark

To strain the network capabilities of the system, the benchmark used a large volume of packet fetches from a random internet address, in this case to google.com. To do this a tool for transferring data using URLs was used called curl.



```
//This code is adapted from code given by the curl API
void benchmarkComm(void) {
    int numPackets = 700;

    for(int i=0; i<numPackets; i++) {
        CURL *curl;
        CURLcode res;

        curl = curl_easy_init();
        if(curl) {
            curl_easy_setopt(curl, CURLOPT_URL, "http://google.com");
            curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);
        res = curl_easy_perform(curl);
        if(res != CURLE_OK)
            fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));

        curl_easy_cleanup(curl);
        }
    }
    return;
}
```

Figure 14: Network communication benchmark

4.2 Hint Results

All tests were completed on a reserved node on the CAC Frontenac cluster with the parameters given in Table 2.

Table 2: Node features

Model	Speed	Cores	Memory
E5-2650 v4	2.7 GHz	24	256 GB

The results for this project were compiled by running ten to twenty iterations of each benchmark, both with and without the respective hint behaviour active, and then averaging the results. What follows is these values, and the ratio between the benchmark run with and without the hint behaviour to the gauge overall effectiveness of the behaviour.

Serial Hint

The serial hint behaviour saw the greatest energy reduction, with the hint behaviour using just about 61% of the energy used by the default case. Despite having no meaningful reduction to runtime, with an energy reduction this significant this is a still a highly promising result.

Table 3: Serial hint results

	Runtime (s)	Energy Usage (J)
Serial hint	20.74	411.14
Default	20.86	672.24
Ratio	0.9942	0.6116

Parallel Hint

The parallel case was very similar as the default case, with no meaningful improvement in either runtime or energy usage. It is interesting though that by increasing the minimum core frequency it still has a slightly lower energy usage. This indicates that by providing a shorter range of frequencies it gives some slight benefit by virtue of the fact that the governor does not need to manually scale up the frequency itself based on demand, which wastes time and energy.

Table 4: Parallel hint results

	Runtime (s)	Energy Usage (J)
Parallel hint	31.18	833.69
Default	31.25	842.83
Ratio	0.9978	0.9892

Memory Bound Hint

The memory bound case saw a slight improvement in runtime of around 1.5% and a slightly better improvement in energy usage of around 4%. While these results are not large, they are significant enough to mention since it shows that this approach does yield some benefits.

Likely in a case that is more heavily memory bound this could see better results since it would be able to decrease the maximum core frequencies more. Although in that case perhaps the memory of the system should simply be improved if it is bottlenecking the system to that degree.

Table 5: Memory bound hint results

	Runtime (s)	Energy Usage (J)
Memory bound hint	4.70	135.43
Default	4.78	140.96
Ratio	0.9833	0.9607

I/O Hint

The results for this case, similarly to the parallel case, lacked any meaningful improvement. This may simply be a function of a benchmark that was not pushing the I/O capabilities hard enough. An idea supported by the fact that for all the behaviours tested for this case, runtime and energy consumption scaled oppositely, which suggests that the CPU is still the limiting piece of hardware in this system. Regardless, there is room for further development with this case.

Table 6: I/O hint results

	Runtime (s)	Energy Usage (J)
I/O hint	16.41	686.12
Default	16.51	688.31
Ratio	0.9939	0.9968

Network Communication Hint

Unlike any of the other hints the communication hint saw greatly improved runtime, running around 30% faster than the default case. It also saw an improvement in energy usage with a roughly 2.3% reduction to energy consumed.

Table 7: Network communication hint results

	Runtime (s)	Energy Usage (J)
Communication hint	1.30	4077.32
Default	1.85	4198.17
Ratio	0.7027	0.9712

4.3 Overall Results

A summary of the reduction ratios from the above tables are graphed below in Figure 15. The results across all the hint cases varied, but in every case, there was at least some level of benefit found by introducing the hint behaviors. Even if the improvement in each case is not massive, if a given program contains sections of each type of computation, the benefits of the impactful hint cases will carry over and lead to overall better program performance.

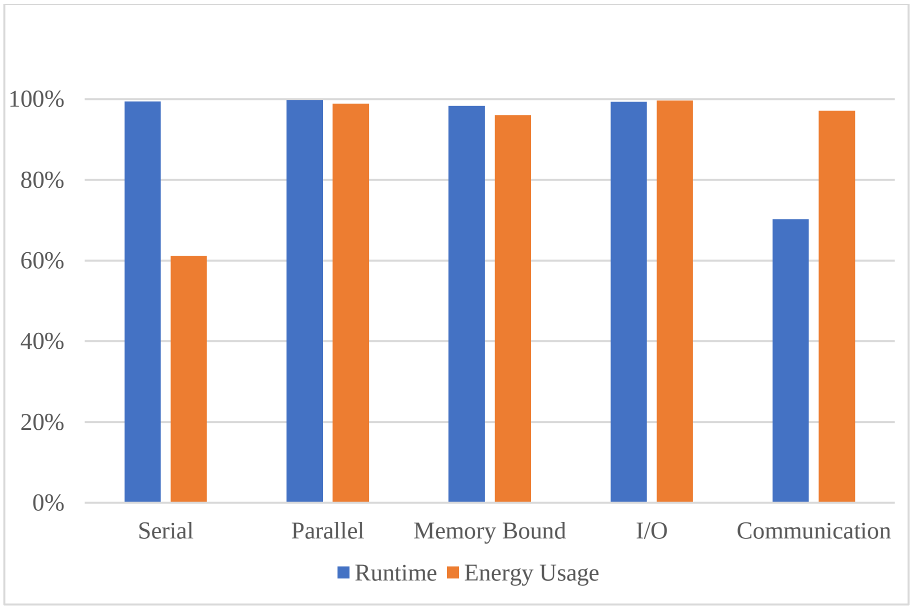


Figure 15: Reduction of runtime and energy usage achieved by hint behaviour

5. Considerations

As with any project, the wider impacts of the undertaking must be carefully considered. There exist stakeholders who may take interest in the design and results of this extension of the Power API. As a community-led open-source project, it is crucial to assess the potential implications of the project's design and results, including robust digital safety measures and considerations for physical safety. Though this project is open-source and therefore free for use with credit, there are aspects of cost surrounding implementation and training that must be considered for feasibility by any HPC group looking to utilize the API. Finally, all reasonable ethical concerns must be addressed, though being quite minor to this project.

5.1 Stakeholders

Several groups may have a stake in this project. The first stakeholder is research groups which utilize HPC resources; if successfully deployed wide scale, they would see the efficiency of the code they deploy increase. However, this would necessitate these groups also learn how to effectively utilize the Power API. While it can be assumed that research groups using HPC resources will have a degree of technical competency, it cannot be assumed that they are knowledgeable in low-level architecture, power systems, or any one specific coding language. Therefore, to best satisfy these stakeholders, who's pressure and preferences may impact acceptance of the Power API into large scale systems, it is important that the developments made in this project are eventually easy to apply, and can be used in as many environments as possible.

Individuals who directly research high-performance computing are counted among stakeholders as well, though with differing interests. The statistical results between CPU power behavior and energy efficiency produced by this project may be of far greater interest than the specific implementation. It can be more safely assumed that HPC researchers are more technically proficient with low-level architecture. To best serve the interests of this stakeholder group, it is important that the gathered data is clear and thorough.

Administrators of HPC systems must also be accounted for as stakeholders. Their concerns lay primarily in safety and ease of use. It is important that any addition to a system is failsafe, and does not create any unnecessary vulnerabilities. The safety of this project is discussed further in 5.2 Safety. Of additional interest to HPC administrators is ease-of-implementation and ease-of-use. Ease-of-implementation is important because, despite likely having a high degree of technical knowledge, time and effort are not unlimited. Ease-of-use may encompass the administrator making modifications to the implementation of the Power API to suit their own needs. However, even ease-of-use on a user level equally matters, as a system administrator is likely to be used as a knowledge resource in an organization. Therefore, to reduce the effort expended on installation and education it is important to maximize ease of implementation and use.

5.2 Safety

As this extension to the Power API interfaces with the operating system on a low level, and has the power to alter power settings within hardware, safety is an important concern. Additionally, the type of HPC systems this code may run on can be incredibly expensive, emphasizing the importance of avoiding hardware damage. With pre-existing guardrails and additional precautions the risks can be minimized.

The first safety concern comes from the alteration of CPU core frequencies. By altering them beyond manufacturer specifications, or even within specifications but higher than usual, there is a risk of overheating, melting, or other various forms of thermal or electrical failure. `Intel_pstate,` through which all operations of this project are run, is on its own quite safe. It takes the upper and lower bounds of frequency set by the `userspace` governor as a suggestion rather than law. It would likely not allow anything truly dangerous to occur.

Two additional risk through misuse of CPU core frequency settings also exist: shortening of hardware lifespan, and hampering performance if used incorrectly. Firstly, hardware is naturally subject to strain from thermal load. To minimize thermal load it is important that the more intense states – namely Serial and Parallel – are only used when intended. Secondly on misuse, though very little can be done to stop an ill-informed developer from declaring a parallelized section of code as serial, steps are taken to prevent such a mistake from impacting other users. Both of these risks are mainly present in the event that the CPU frequencies are altered and then never reverted to their original state. This is addressed in how the Power API, and this extension, function: at the beginning of a hint region the power state of the CPU is read and saved; then, once the hint region ends the API restores the CPU to its previous state, reinstating any settings or governors. This prevents any changes made by the API from “leaking”, potentially causing harm to slowdown. There is still the risk of an application using the Power API crashing part-way through a hint region, in which case the API would not have the opportunity to restore the CPU state. In this case, there is a script, `reset.sh`, that restores all settings.

5.3 Costs

The Power API is a community-led open-source project, meaning there is not monetary cost associated with development of the project, and no purchase required for any user or HPC looking to utilize it. While the Power API offers immense value in terms of its capabilities, it's important to note that there may be potential costs and savings associated with utilizing it in a more refined state in the future. As highlighted in the 5.1 Stakeholders section, one of the barriers to entry for users is the installation process and the learning curve associated with understanding its functionality, which may require effort and time from system administrators. However, this initial investment can be considered a one-time cost in the form of salary and effort, which can be easily offset by the substantial savings that can be achieved by leveraging the Power API.

One of the primary advantages of utilizing the Power API is the potential for significant electricity usage reduction, resulting in cost savings in the long run. By providing insights and control over power consumption, the Power API empowers users to optimize their energy usage and minimize wastage, leading to potential cost savings on electricity bills and reduced environmental impact. This not only benefits the immediate users of the Power API but also has a positive ripple effect on the broader community and the environment as a whole. Environmental impact is further discussed in 5.4 Ethics.

5.4 Ethics

While this project aims to be a beneficial tool for improving CPU efficiency in high-performance computing systems, there may exist a few ethical concerns related to data privacy and transparency. For instance, the collection and utilization of power usage data may raise privacy concerns if not handled appropriately, as it could potentially reveal sensitive information about system usage patterns or user behavior. Ensuring that the data collected is securely maintained ultimately falls to the individual HPC groups using the Power API. While this extension does log power data, and other portions of the Power API specialize in data collection the original documentation states that the Power API "assumes that the Resource Manager has a data retention capability (database) available to query energy and statistics information based on job or user Id. The availability of this information is implementation dependent." [1].

Concerns regarding transparency are understandable, especially around an API with the ability to control hardware on a low-level. However, this and nearly any other concern can be dismissed by the community-led nature of the project. The source code of the Power API is visible to anyone, meaning anyone can independently verify the safety and integrity of it. Furthermore, the group behind the Power API at Sandia National Laboratories welcomes community contributions.

6. Compliance with Specifications

Table 8: Specifications table

1	**Functional requirements**	**Specification met?**
1.1	Regulate a device's power controls with the Power API.	Yes
1.2	Integrate with the Power API framework and structure.	No
1.3	Enable compatibility with Intel and AMD devices.	Yes
1.4	Implement default region hint power control behaviour.	Yes
1.5	Implement serial region hint power control behaviour.	Yes
1.6	Implement parallel region hint power control behaviour.	Yes
1.7	Implement computation region hint power control behaviour.	No
1.8	Implement network communication region hint power control behaviour.	Yes
1.9	Implement input/output region hint power control behaviour.	Yes
1.10	Implement memory bound region hint power control behaviour.	Yes
2	**Interface requirements**	

2.1	Enable the defined hint behaviours to be compatible with the Power API suite of hint functions (PWR_AppHintCreate(), PWR_AppHintDestroy(), PWR_AppHintStart(), PWR_AppHintStop(), PWR_AppHintProgress()).	No
2.2	Provide debug information to display what power controls are being changed.	Yes
3	**Performance requirements**	
3.1	Each region hint's power control behaviour will reduce the power consumption when used on an appropriate region of code. No specific quantitative values can be stated as a key aspect of this project is determining what reductions can be achieved.	Yes
3.2	Each region hint's power control behaviour will reduce the runtime of a program when used on an appropriate region of code. No specific quantitative values can be stated as a key aspect of this project is determining what reductions can be achieved..	Yes

During the initial phases of this project, a series of requirements were established to guide the development process, as outlined above in Table 8. The team has made significant progress in meeting these requirements, with most of them being successfully fulfilled. However, there are a few exceptions where certain requirements have not been fully met or may require further attention. It's worth noting that the importance of each requirement may vary, and not all requirements carry the same weight in terms of their impact on the overall project.

The team successfully updated a devices power settings, established control behavior for several hint types, and provided measurable reductions to energy usage. There was also success in providing some support for AMD and Intel devices, though most of the research and testing was done on Intel devices. Regardless, the information gathered through this project will have some across architectures.

The first, and more minor, requirement that was not fulfilled was the research and implementation of the compute hint region. Early on in project scoping it was established that the compute hint should be of lowest priority, since in any situation where it could be used, the serial or parallel hints could be used to similar effect. Either way, future work may be undertaken to accommodate the compute hint region.

The second, and more substantial requirement that was not met was integration with the Power API, as noted in requirement 1.2 and 2.1. Part-way through the project it was realized that the most viable path would be to develop benchmarks in C with batch files to adjust the power settings and postpone Power API integration. Time requirements have unfortunately moved Power API integration into the scope of future work. While the Power API integration was not completed as initially planned, this update to the project scope was deemed a reasonable adjustment, as the primary focus was on achieving meaningful research results. The team acknowledges that Power API integration remains an important area for future work to further enhance the capabilities and functionalities of the project. While the research results achieved through this project are significant, integration with the API remains the path to providing the most benefit to general HPC users and researchers alike.

7. Conclusion

The stated goal of this project, to improve runtimes and reduce energy usage with the use of region hint behaviour, was achieved. Some of the cases exceeded expectations, such as the serial and network communication cases, and showed massive reductions in energy usage and runtime respectively. While others such as the parallel and I/O cases, barely saw any improvement at all. Realistically likely neither of these cases represent what can be expected from the introduction of region hints into HPC jobs. Instead, it will likely be more similar to the memory bound case, with a reduction of 1.7% to runtime and 4% to energy which will be reliable attainable. Regardless of the magnitude, this project proved to show the potential of a consistent, if humble, improvement in runtime and energy usage with the introduction of hint behaviour and dynamic frequency scaling. Below will be a synthesis of the methods in which the discussed reductions were achieved.

7.1 Key Takeaways

Serial optimization

Any situation which primarily recruits only one of the available computational devices has the potential to see improved runtimes and/or reduced power consumption using dynamic frequency scaling. With this project both the serial and network communication hint cases showed significant improvement by increasing a single core's minimum frequency and decreasing the other core's maximums.

Minimum/maximum frequency boundaries

If a core is likely going to be operating in a certain frequency range, providing tighter minimum and maximum bounds appears to have some improvement. To what degree these bounds can be set depends for what time the core will likely be within that range, how the governor scales the frequency within that range, as well as some thermal constraints at higher frequencies. These improvements are likely from inefficiency in the governor updating the frequency, as depending on which governor is active it may be slow to track demand and bring the core frequency either up or down.

Non-CPU-bound frequency reduction

When the runtime of a program is bound by another subsystem or device other than the CPU, energy can be saved by lowering the core maximums down to ideally just above the frequency at which the other device is able to be receiving from or responding to the CPU. Finding this bound will be difficult as it is on a per program basis, but there may be room for future development to dynamically track this.

7.2 Recommendations

Integrate with the Power API

The work done for this project is intended to be used for the development of the Power API. If this work could be integrated with the existing API it would allow the benefits discovered in this project to be fully realized. Before this happens, two other elements need to be done. Firstly, the computation hint case should be implemented, and secondly, more testing should be done on non-Intel devices to ensure these findings stand across multiple architectures.

Tight min/max frequency bounds

The best performing cases were those in which one core’s minimum was boosted at be around 80% of it’s maximum, and the other cores maximums were reduced to just above their minimums. These cases may be able to be improved further if a tighter minimum/maximum range can be set. Doing so was not done in this project due to thermal concerns about running the cores at such high speeds for prolonged periods, however, if it could be done safely, the serial and network communication hints may see even better runtimes.

Non-CPU-bound frequency setting

As discussed above, for programs or regions that are memory bound or otherwise bound by a device other than the CPU, it may be beneficial to find a way to scale the CPU core maximum frequencies down to a point before it becomes CPU bound again. Any static ratio will fail to be completely general to all regions of that type, so a more complex approach will be required to maximize the efficiency in these regions.

7.3 Impact

The most immediate impact of this project is the continued development of the intelligence of CPU governor systems, both automatic and manual. Using the data gathered, future work can focus on refinement and improved implementation for the purpose of bettering the field of dynamic CPU frequency scaling, as well as HPC systems as a whole.

Furthermore, this project has the capability to have far reaching positive environmental impacts. As a frame of reference, the Japanese HPC system known as Fugaku used, on average, 270 megawatt hours per day, based on their Operation Status page [3]. Our most promising results, in the Serial case, used 39% less energy, however it cannot be assumed that all code will be serial, and reductions were much less significant in other cases. However, if this extension of the Power API can reduce energy consumption, on average, by even 10%, that would translate to enough energy to continuously power 1000 homes in Ontario, Canada [4]. In terms of carbon emissions, this hypothetical reduction would be 6,987 metric tons of CO_2 per year, just from the one facility [5].

Distinguishing this project from others, as previously discussed throughout the report, is its non-commercial, open-source, community-led nature. This means that these potential energy savings, and by proxy environmental impacts, are in no way locked behind monetization, free for all to use. Furthermore, the technology and techniques used within it are in no way proprietary or secret, meaning future developers and HPC researchers can expand and improve upon the Power API freely, as long as credit is given. The impact of this project does not live and die with it, but in all likelihood will continue to grow as it is adopted and improved upon by anyone with the skill to do so.

8. Effort

Name	Overall effort expended
James Corley	100 %
Simon Dudtschak	100 %
Alex Perrin	100 %

References

[1] R. E. Grant, M. Levenhagen, S. L. Olivier, D. DeBonis, K. T. Pedretti and J. H. Laros III, "Standardizing Power Monitoring and Control at Exascale," in Computer, vol. 49, no. 10, pp. 38-46, Oct. 2016. http://dx.doi.org/10.1109/MC.2016.308

[2] James H Laros, Suzanne M Kelly, Steven Hammond, Ryan Elmore, and Kristin Munch. Power/Energy Use Cases for High Performance Computing. Internal SAND Report SAND2013-10789. https://digital.library.unt.edu/ark:/67531/metadc869854/.

[3] *Operation Status of Fugaku*. [Online]. Available: https://status.fugaku.r-ccs.riken.jp/d/fugaku/0-operation-status-of-fugaku?orgId=1. [Accessed: 13-Mar-2023].

[4] "Residential Electricity and Natural Gas Plans," *EnergyRates.ca*, 01-Sep-2020. [Online]. Available: https://energyrates.ca/residential-electricity-natural-gas/. [Accessed: 13-Mar-2023].

[5] "Greenhouse Gases Equivalencies Calculator - Calculations and References," *EPA*. [Online]. Available: https://www.epa.gov/energy/greenhouse-gases-equivalencies-calculator-calculations-and-references. [Accessed: 09-Apr-2023].

[6] The Linux Kernel Archives. (n.d.). Retrieved April 10, 2023, from https://www.kernel.org/doc/Documentation/cpu-freq/intel-pstate.txt

[7] Mochel, P., & Murphy , M. (2011, August 16). The Linux Kernel Archives. Retrieved April 10, 2023, from https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt

[8] Brodowski, D., Golde, N., Wysocki, R. J., & Viresh Kumar. (n.d.). The Linux Kernel Archives. Retrieved April 10, 2023, from https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt