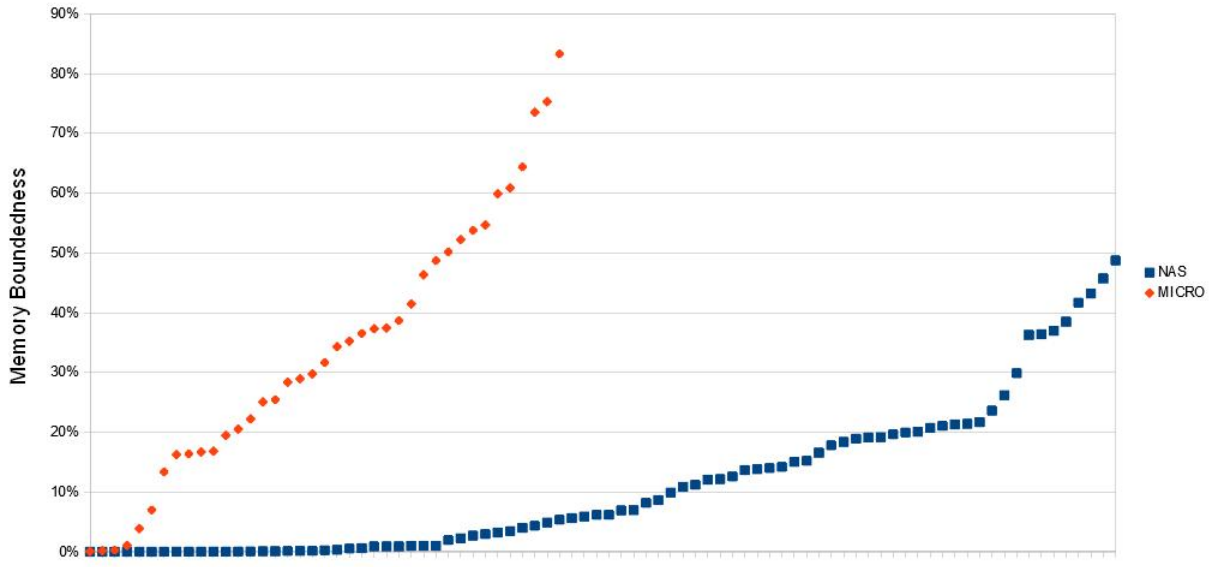
CHALMERS
UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG



Creating a Microbenchmark with wide coverage of Memory-boundedness

Investigating memory-boundedness by creating a microbenchmark to characterize DVFS behaviour

Niklas Côté

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Creating a Microbenchmark with wide coverage of Memory-boundedness

Investigating memory-boundedness by creating a microbenchmark to characterize DVFS behaviour

Niklas Côté

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Creating a Microbenchmark with wide coverage of Memory-boundedness
Investigating memory-boundedness by creating a microbenchmark to characterize DVFS behaviour
Niklas Côté

© Niklas Côté, 2022.

Supervisor: Bhavishya Goel, Department of Computer Science and Engineering
Examiner: Miquel Pericas, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Creating a Microbenchmark with wide coverage of Memory-boundedness
Investigating memory-boundedness by creating a microbenchmark to characterize DVFS behaviour
Niklas Côté
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Memory-boundedness of an application is defined as the degree to which the performance of the application depends on the size and performance of memory instead of the CPU. The degree of memory-boundedness of an application determines its speedup when the CPU frequency is increased: an application with no memory-boundedness will exhibit linear speedup with frequency increase while an application with 100% memory-boundedness will exhibit no speedup at all. Dynamic voltage and frequency scaling (DVFS) is a power saving technique which aims to save energy by dynamically reducing frequency during the memory-bound phase of the application. The DVFS decision making is based on prediction models which predict the appropriate voltage and frequency for the application phase. To increase the accuracy of prediction models, training data needs to be collected from applications which exhibit varying degrees of compute-bound and memory-bound behavior. A single microbenchmark which can simulate wide variations of memory-boundedness behaviour could reduce the time required to train the prediction models and improve prediction accuracy.

This thesis analyzes different measurement methods for memory-boundedness and proposes a new formula based on L3 cache misses which is believed to be a more suited fit for the definition of memory-boundedness. The testing of formulas was done on a benchmark suite from NASA called NAS Parallel Benchmarks (NPB), where the consistency and values of the formulas were evaluated. The result of measurements with the new L3 cache miss formula was then used to create a microbenchmark which can produce large variations of memory-boundedness.

Keywords: DVFS, memory-boundedness, microbenchmark, benchmark, voltage, frequency, prediction models, static voltage, dynamic voltage

Acknowledgements

Most of all I would like to thank my supervisor Bhavishya Goel, for being a great teacher and mentor for this very interesting project and for always believing in me and motivating me to learn more. I also want to thank everyone involved in the presentation and writing process of this project, Miquel Pericas, Marcus Mathiason and Edvin Leidö, in helping me clarify and further improve the thesis. Finally I would also like to thank my partner, my family and my friends for always being there for me and pushing me to be my very best.

Niklas Côté, Gothenburg, July 2022

Contents

List of Figures

List of Tables

1
Introduction

This chapter introduces the project background, the problem statement and the project limitations.

1.1 Background

Dynamic voltage and frequency scaling (DVFS) is the mechanism of tuning the CPU frequency to match the needs of the ongoing execution and tuning the voltage to fit the CPU frequency. This has the potential to decrease energy consumption of systems while maintaining performance within acceptable levels as defined by QoS (Quality of Service). Most modern microprocessors today have the ability to harness the power of DVFS, which can be used to optimize the amount of energy required to run a particular application. In order to maximize the efficacy of DVFS, the optimal voltage-frequency pair has to be chosen in real-time. Many scheduling algorithms that employ DVFS make use of models that can predict the impact of changing the frequency and voltage on performance and energy consumption [3, 6, 11, 9]. The application characteristics like memory-boundedness and concurrency decide which voltage-frequency pair is optimal [2] for energy efficiency. Creating effective DVFS prediction models requires availability of benchmarks with wide-ranging memory-boundedness and concurrency behavior, either to generate the training set or to test the accuracy of the models. A systematic characterization of the impact of voltage and frequency change on the performance and energy consumption over a wide range of application characteristics can be very useful to generate training and/or test data. As part of this thesis, we have developed a parameterizable microbenchmark which can be finely tuned to mimic benchmarks with different memory-boundedness and concurrency behavior. This microbenchmark can be used to collect information about impact of voltage-frequency change on performance and energy consumption across the entire DVFS configuration space of the platform.

The purpose of this microbenchmark is to create deeper understanding of DVFS behavior and characteristics depending on memory-boundedness of applications. This knowledge can be used to create more accurate prediction models in the future. These prediction models rely on already available benchmarks with varying degree of memory-boundedness behavior. The problem is that finding the right set of benchmarks that represent wide range of memory-boundedness can be fairly difficult. As per the data collected by computer architecture research team at Chalmers, 40 different benchmarks from NAS benchmark suite [4] were required to collect data

depicting wide enough memory-boundedness behavior for training the DVFS prediction model. Apart from being time consuming, it is also difficult to debug the prediction model using such training data since the memory-boundedness cannot be changed orthogonally to the rest of the benchmark behavior. A microbenchmark that can be finely tuned to exhibit desired degree of memory-boundedness would be a great tool to do systematic and comprehensive study of DVFS characteristics of the platform so that more accurate DVFS prediction models can be created. To the best of our knowledge, such a microbenchmark to explore DVFS configuration space does not exist.

1.2 Problem statement

Currently there are no tools to exclusively express different memory-boundedness behaviours, one would have to use a number of different benchmarks to do so. Different benchmarks when compared can vary in other features than solely memory-boundedness and is therefore not ideal in our case. Large variations in benchmarks create difficulties for researchers to compare and evaluate memory-boundedness behaviour. The purpose of this thesis is to solve this issue by creating a microbenchmark tuned for various memory-boundedness behaviours without changing larger parts of the application. Such a tool can lead to deeper understanding of DVFS behavior and help prediction models to increase accuracy over application phases that exhibit varying degrees of memory-boundedness behaviour.

1.3 Limitations

For the purpose of this master thesis, we have limited our experiments and data collection to just a single platform although the microbenchmark and the methodology should be portable to other platforms. The tests were conducted on a limited subset of available settings to reduce the experimentation run time. The scope of this thesis was limited to creating the microbenchmark and comparing the characteristics of microbenchmark with those of NAS benchmark. The training and/or testing of existing DVFS prediction models using the created microbenchmark can be done as part of future work.

2
Theory

This chapter contains an overview of DVFS, frequency and voltage. Techniques to measure memory-boundedness and creating a microbenchmark are also presented. Lastly the reasoning and explanation of the chosen methodology to create the microbenchmark is also covered.

2.1 Frequency

The frequency of a CPU is the speed of the clock, which determines how quickly instructions are being processed, which means that a higher frequency leads to a higher performance of the CPU. The CPU frequency has been a measurement to compare CPUs, but in the age of multi-core CPU systems it is no longer the only measurement to compare CPUs. Most CPUs today have similar maximum, minimum and average frequency, but have very different amount of instructions per cycle (IPC). The amount of instructions which can be executed per cycle have been increased in modern CPUs by utilizing computer architectures such as multi-core, many-core and various memory scheduling protocols for those architectures. The new techniques mentioned utilize computer architectures to increase performance of CPUs which is more beneficial to increase performance compared to increasing the frequency. The frequency speed is limited to the speed which transistors can transfer signals and the transistors are limited to the capacitance of the material used to create them. Since the capacitance for transistors has not increased as fast as the increase of frequency, increasing frequency further would lead to higher CPU temperatures and inefficient energy usage.

Efficient frequency utilization plays a large role in decreasing the energy consumption of all computational units. If the frequency is increased and instructions are not ready to be processed, the increase in frequency is not utilized efficiently and only leads to larger energy consumption and temperature of the CPU. Changing the frequency to fit the needs of the execution can lead to large energy savings and an increase in the life-span of computational components.

2.2 Voltage

The voltage in a CPU is provided by the power source and leads power to the transistors to have the ability to transfer signals. As the voltage increases, the transistors are able to send signals faster due to more intensive electric fields. The voltage of

a CPU has limits since increasing the voltage leads to higher power leakage, which increases the temperature of the CPU. When the temperature of a CPU increases too much, the efficiency and durability of the transistors starts to be affected negatively, resulting in system failures and slower execution times.

Voltage provides power to the CPU and is a requirement to compute instructions. To achieve efficient power savings the voltage needs to be adjusted to the frequency of the CPU. When the frequency is lowered the voltage should then be decreased to achieve larger power savings, since the transistors does not need to send signals as fast. The calculation of power usage is a well known formula which depends on the combined power usage of the dynamic portion and the static portion, with the variables capacitance, voltage, static power leakage and frequency of the CPU. The formula 2.3 shows that the energy usage decreases as voltage and frequency lowers. However the energy efficiency may also decrease since more total energy is required to compute the same amount of instructions. To achieve efficient energy usage the voltage and frequency need to be balanced with performance and execution time.

$$\text{Static Power} = I_{leakage}V \tag{2.1}$$

$$\text{Dynamic Power} = \alpha CV^2 f \tag{2.2}$$

$$\text{Total power} = (\alpha CV^2 f) + (I_{leakage}V) \tag{2.3}$$

2.3 Static and Dynamic voltage

The voltage can be static or dynamic but that does not change the effects of increasing or decreasing voltage. Static voltage is when the voltage is static and does not change during execution or idle state of the CPU. Dynamic voltage has the ability to change the voltage depending on the work-load of the CPU and the frequency, to decrease energy consumption and maintain high performance.

2.3.1 Static voltage

Static voltage is when the voltage stays at a certain level for the entire execution even if the CPU is idle. The energy usage of static voltage is determined by the voltage and the leakage of power which is affected by temperature and transistor voltage. To limit the energy consumption when using static voltage, leakage of current, voltage and temperature of the CPU need to be limited 2.1.

2.3.2 Dynamic voltage

Dynamic voltage is when the voltage switches the amount of supply voltage to the CPU, preferably to fit the work-load and frequency which leads to energy savings by not consuming unnecessary resources. The energy consumption of dynamic voltage is calculated by the capacitance of transistors in the CPU, the voltage and frequency.

To limit energy consumption using dynamic voltage, voltage and frequency need to be limited 2.2.

2.4 Dynamic voltage and frequency scaling

There are many techniques to adjust the CPU frequency and voltage to fit the needs of the system. Techniques such as over-clocking, under-clocking and DVFS are used to fit the user or application requirements on performance and energy. One large benefit with DVFS compared to over- and under-clocking is that DVFS is dynamic and not static during run-time, making it able to choose frequency and voltage depending on the needs of the execution and allocates necessary resources. The DVFS operates by decreasing the CPU frequency and voltage when the CPU is idle or has a low workload, and increasing it when there is a higher workload. Thereby utilizing energy, performance and resources more efficiently when using DVFS. The DVFS decision making is controlled in the operating system by using DVFS prediction models. Prediction models can be specific for types of system and application characteristics. To achieve higher accuracy of the DVFS decisions, the operating systems needs to be able to predict and schedule changes of frequency and voltage. The scheduling and prediction of DVFS is a widely researched area. The reason for this is the large variety of tasks and systems DVFS can be optimized for. Prediction and scheduling models are often more accurate for certain systems and workloads, such as memory-bound or compute-bound workloads. DVFS changes the CPU-frequency and voltage to save energy, but DVFS does not change the frequency in memory. The memory-boundedness of applications complicate the extent of usefulness and accuracy of DVFS scheduling.

2.5 Defining Memory-boundedness

Memory-boundedness is a characteristic which can be evaluated and measured by using various microbenchmarks and benchmarks. Memory-boundedness has been measured differently in various papers such as [2], [8] and [9], but essentially memory-boundedness is defined as the degree of work which is stalled by memory. Each execution has a portion of the work-load in memory and in the CPU. What is special about the memory-bound part is that when changing the CPU frequency the stall time in memory does not scale with the CPU portion. So an increase in CPU frequency scales the execution time for the compute-bound part, but the execution time of the memory-bound part stays the same. The memory-bound portion becomes a bottleneck in decreasing execution time when increasing CPU frequency, as can be seen in figure 2.1. The calculations of memory-boundedness vary between papers since they measure different aspects of the memory-boundedness but figure 2.1 shows the basic concept of why memory-boundedness affects the degree of efficient frequency utilization.

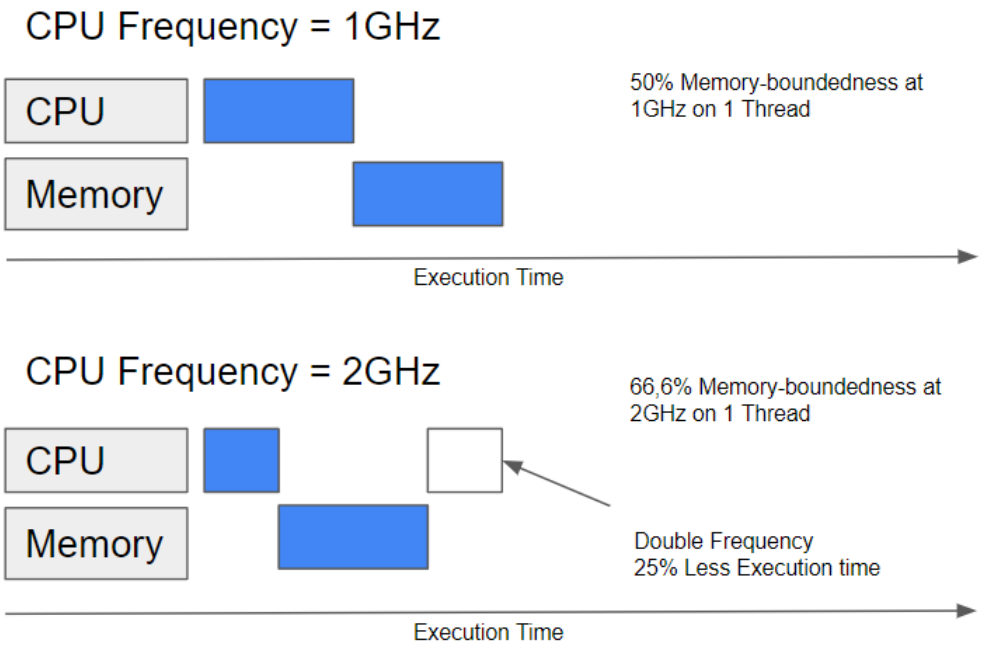


Figure 2.1: Shows how the compute-bound portion scales linearly with frequency and how the memory-bound portion does not scale with frequency.

2.6 NAS benchmark suite

The NAS Parallel Benchmarks (NPB) is a benchmark suite from NASA for testing high-performance systems and supercomputers. The NPB is an open source platform consisting of various benchmarks with various system characteristics. NASA provides, updates and shares these benchmarks with researchers and the public to test the performance of many systems. The benchmark suite contains a variety of different benchmarks with different characteristics, working-set-size and has the ability to run in parallel. The benchmarks simulate different types of work-loads, which researchers use to investigate system behaviour and optimize their systems for these types of applications.

The version used in our thesis is NPB 3.4.2 with the programming model OpenMP. This version contains benchmarks IS, EP, CG, MG, FT, BT, BT-IO, SP, LU, UA, DC and DT, but only versions IS, EP, CG, MG, FT, SP, LU and UA will be used in our thesis. The motivation for only using these is the variation of memory-boundedness the other benchmarks exhibits is already covered from the selected benchmarks.

List of NAS benchmarks tested

- IS - Integer Sort, random memory access
- EP - Embarrassingly Parallel
- CG - Conjugate Gradient, irregular memory access and communication

- MG - Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive
- FT - discrete 3D fast Fourier Transform, all-to-all communication
- SP - Scalar Penta-diagonal solver
- LU - Lower-Upper Gauss-Seidel solver
- UA - Unstructured Adaptive mesh, dynamic and irregular memory access

2.7 Creating a Microbenchmark

Benchmarks are programs created to simulate a workload in order to evaluate and measure performance of a system. Microbenchmarks are benchmarks used to simulate a specific single system characteristic, often in the hardware at the lower-level. Microbenchmarks are widely used by researchers to gain a deeper understanding of system behaviour in hardware, since microbenchmarks can be configured. Performance counters are often used to measure the system behaviour when running a microbenchmark. Depending on the data collected by the performance counters, the microbenchmark can be optimized further to generate the wanted system behaviour.

The system behaviour of interest is various range of memory-boundedness. To create a microbenchmark with wide range of memory-boundedness, instructions to produce memory stalls and instructions to avoid memory-stalls are required. The instructions used to generate memory stalls are instructions which generate L3 memory misses and L1-, L2- or L3-cache hits, the instructions to avoid memory stalls are simple additions and multiplications of values.

3
Related work

This chapter contains related work on how to measure memory-boundedness and using the knowledge of memory-boundedness behaviour to train prediction models.

3.1 Measuring Memory-boundedness

Goel et al. [2] calculates the memory-boundedness of an application by dividing the execution in two parts, a compute-bound part and a memory-bound part. The execution part is the part which scales execution time depending on the CPU frequency, i.e. the part which does not scale with CPU frequency is the memory-bound part of the execution. The variables to calculate the compute-bound and memory-bound part of the execution are frequency and execution time at two different frequencies, as can be seen in equation 3.1. It is a formula which is based on theoretical assumptions, but in practice is difficult to measure accurately. The memory-boundedness could vary for one frequency depending on which other frequency is used to calculate memory-boundedness. For example, if calculating memory-boundedness for frequency 4 Ghz, the memory-boundedness is different if you use 2 Ghz or 3 Ghz as the second frequency in the formula. The memory-boundedness results of an application running on a frequency should by the formula be the same no matter which frequency is used, but in practice is not the case, which makes it unreliable.

$$T_F2 = T_F1 * (M + (1 - M) * \frac{F1}{F2}) \tag{3.1}$$

$M =$ The fraction which the CPU does no work, stalling for memory

$T_Fx =$ Execution time at frequency x

$Fx =$ Frequency at setting x

The equation 3.1 shows how Goel et al. calculates the memory-bound part of an execution by scaling the frequency (from F1 to F2) and measures the difference in execution time.

Spiliopoulos et al. [9] measures memory-boundedness as the total stalls divided by number of cycles, as can be presented by equation 3.2. Their formula is overall simple to use, however a downside with the formula is that the total stalls may also include execution based stalls and could include stalls that are dependent on the CPU and not memory. In the tests that were supposed to give results approximately of 0% memory-bound (very low amount of memory accesses), the formula 3.2 produced

results of the execution being around 15% memory-bound even if very low amounts of memory access was measured.

$$Memory - boundedness = \frac{Total\_Stalls}{Cycles} \tag{3.2}$$

Molka et al. [8] measures the memory-boundedness as the max out of stalls caused by L1 cache data misses or resource stalls, then divide that by the number of cycles. It is a consistent formula but gives a underestimation of the memory-boundedness since L1 cache scales with the CPU frequency. Memory-boundedness should by our definition not scale with the CPU frequency.

$$\text{Stalls in Store Buffer} = RESOURCE\_STALLS : SB$$

$$\text{L1 Data Cache Misses} = CYCLE\_ACTIVITY : STALLS\_L1D\_PENDING$$

$$Memory - boundedness = \frac{MAX(SB, STALLS\_L1D\_PENDING)}{Cycles} \tag{3.3}$$

With the inspiration from the previous formulas and the definition used in prediction models, the decision to create a new formula was made. The formula will be easy to use, measure and understand. It should be accurate and not overestimate the degree of memory-boundedness. The formula is based on L3 cache misses, misses in L3 cache causes long memory stalls which are highly significant to the execution. L3 stall cycles are the most significant time consuming activity of the memory. Memory-boundedness should be the part of the execution which does not scale with frequency of the CPU. Assuming that the core and uncore frequency are in sync, the L1, L2 and L3 caches are affected by changes of the frequency of the CPU, misses to those caches are by definition not fully memory-bound. The misses on the L3-cache are fully memory-bound since each miss is waiting for memory (DRAM) which is not affected by the CPU frequency. The L3-miss performance counter will in theory create an accurate estimate of the memory-boundedness, since it calculates the misses which L3 cache stalls waiting for memory. The formula can be seen in figure 4.1.

$$Memory - boundedness = \frac{CYCLE\_ACTIVITY : STALLS\_L3\_MISS}{Cycles} \tag{3.4}$$

3.2 Prediction models

Prediction models are based on statistical data from running benchmarks with varying characteristics, to predict the upcoming optimal voltage-frequency pair for a program with unknown characteristics. There are several ways to create prediction models and the topic is widely researched.

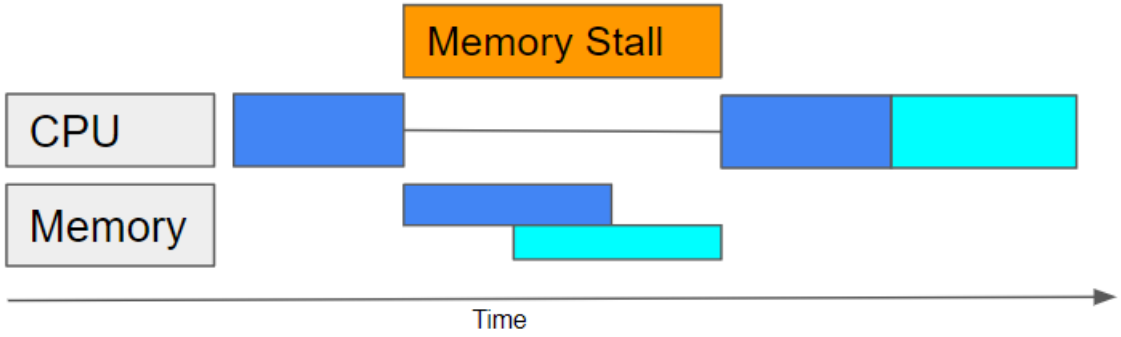


Figure 3.1: Stall based prediction model assumptions, shows how the CPU still stalls even if work is ready to be done on CPU.

3.2.1 Stall based

The Stall based prediction model [1], [5] is a simple model which makes it widely usable in various DVFS systems. It is based on performance counters and the assumption that when a memory-stall occurs the CPU becomes fully idle. This approach leads to low accuracy rate in out-of-order pipelines, since other instructions could in practice still execute during a memory-stall. Overlapping misses can also lead to even longer stall time. If the previous miss returns and could be executed on the CPU, it will not if there is another miss stalling the CPU. The strength of this model is however the simplicity and the DVFS system usability.

3.2.2 Leading Loads

The leading loads prediction model [1], [5], [10] was developed around 2010 and was created as an improved version of the stall based model. The difference compared to the stall based model is the understanding of how out-of-order pipelines can exploit memory-level parallelism, to not overestimate the amount of stall time. Leading loads took into account that several memory-misses can occur in parallel and overlap in out-of-order pipelines. Leading load is the time of the first memory stall in a burst of several memory misses. The rest of the stall time for other misses is ignored. When the leading load finishes, the next memory stall will be the new leading load. By understanding memory-level parallelism and the occurrence of overlapping misses, the leading loads approach managed to improve the accuracy of DVFS prediction models in many different systems. The leading loads was also simple enough to be implemented on real hardware just as stall based, but with higher accuracy in DVFS prediction. The disadvantages of the leading loads approach is the lack of tools, to evaluate memory system behaviours such as prefetching, memory-locality and memory scheduling policies.

3.2.3 CRIT

The CRIT prediction model [7], expands further than leading loads on tools for evaluation of the functionality of memory systems. CRIT uses an algorithm to

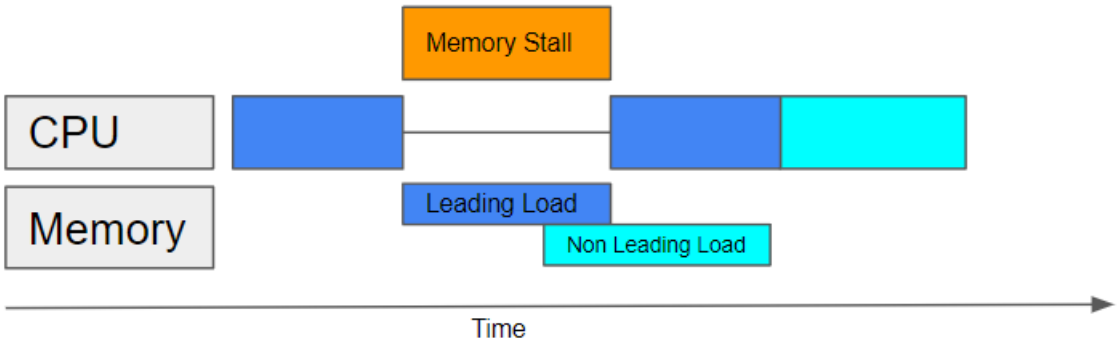


Figure 3.2: Leading loads prediction model assumptions, shows how the CPU benefits from being able to do work when ready, compared to 3.1.

determine the memory stall time of dependent misses, which means the misses that contains necessary data for the CPU to continue execution. Which is an more accurate assumption compared to leading loads, since a leading load could be non-dependent and the CPU could execute and thereby benefit from an increase in frequency.

4
Method

The method of this thesis will be covering how to define memory-boundedness and tools used for measuring it. Then the structuring, implementation and testing of the microbenchmark is covered, also the tuning of various settings.

4.1 Measuring Memory-boundedness

Memory-boundedness is defined as the part of the execution time which does not scale with the CPU frequency. Previous work([2], [9]) has shown that the memory bound fraction of the application can be approximated by measuring the CPU stalls caused by memory accesses.

Measuring the theoretical memory-boundedness requires isolating the CPU stall cycles caused specifically due to memory accesses. However in modern out-of-order processors, multiple stall events (branch misprediction, RS/ROB stalls, cache misses, etc) may happen in parallel with memory access stalls making it very challenging to isolate stalls due to one specific type of event. Therefore we started the testing and measurements of memory-boundedness with all formulas 3.1, 3.2 and 3.3 mentioned in chapter Theory, except 4.1 which was added and created later on. To investigate which formula to use for creating the result for NAS benchmark suite of memory-boundedness coverage, the formulas were tested for the EP and MG benchmarks on frequencies 800MHz, 2GHz, 3GHz and 4,4GHz, and number of threads 1, 2 and 4 in tables 4.1 and 4.2.

The results in the tables are a comparison between a benchmark with almost no memory accesses and a benchmark with high amounts of memory accesses. To investigate the formulas, the behaviour and consistency was used as evaluation. The behaviour for the formulas was similar when increasing frequency and number of threads. From these formulas, 3.1 varied the most and was not consistent or simple enough in our case. The formula 3.2 had a strange value for the EP benchmark since it resulted in a 15% memory-boundedness for a benchmark which should be close to 0%, since the execution time of the EP benchmark scaled almost perfectly with the CPU frequency. The results of 3.2 can only be explained by that the performance counter STALLS_TOTAL also includes some sort of execution stalls in the calculation. Since that is affected by changing the frequency of the CPU that is not an accurate measurement for memory-boundedness. The formula 3.3 looks more promising because of the simplicity to measure and evaluate but also the consistency.

Table 4.1: Embarrassingly Parallel benchmark (EP), high intensity of computational instructions, memory-accesses close to none. Expected result around 0% memory-boundedness. *Negative values and values above 100% are considered errors, since that occurrence is not possible (*).*

Thread 1	**0.8GHz**	**2GHz**	**3GHz**	**4.4GHz**
Goel-Formula	-0.60%*	-0.20%*	0.10%	-0.30%*
Spiliopoulos-Formula	15.10%	14.90%	14.90%	14.90%
Molka-Formula	0.20%	0.20%	0.20%	0.10%
L3-Stall formula	0.00%	0.00%	0.00%	0.00%
Threads 2	**0.8GHz**	**2GHz**	**3GHz**	**4.4GHz**
Goel-Formula	-0.90%*	-0.40%*	0.80%	-1.10%*
Spiliopoulos-Formula	15.10%	15.00%	15.00%	14.90%
Molka-Formula	0.20%	0.20%	0.20%	0.10%
L3-Stall formula	0.00%	0.00%	0.00%	0.00%
Threads 4	**0.8GHz**	**2GHz**	**3GHz**	**4.4GHz**
Goel-Formula	-0.10%*	0.00%	-0.10%*	0.20%
Spiliopoulos-Formula	15.00%	15.00%	15.00%	15.10%
Molka-Formula	0.20%	0.20%	0.10%	0.10%
L3-Stall formula	0.00%	0.00%	0.00%	0.00%

However those counters will produce an overestimation of how many instructions are memory-bound, since they include stalls which are affected by frequency of the CPU.

The later created formula 4.1, uses a newly added Intel specific performance counter to calculate the total L3-misses stall cycles of all cores (CYCLE_ACTIVITY:STALLS_L3_MISS). When the L3-cache misses it stalls to memory, which in theory does not scale with the CPU frequency unlike L1, L2 and L3 cache accesses. The L3-miss formula 4.1 was based on the assumption that memory-boundedness is the part which does not scale with CPU frequency and is the most consistent and simplest formula. The formula 4.1 was the one we used to create the results of memory-boundedness of NAS benchmarks and later on the microbenchmark.

4.2 Microbenchmark

The microbenchmark is created to have a wide range of memory-boundedness with the formula 4.1. To measure memory-boundedness the performance counters CYCLE_ACTIVITY:STALLS_L3_MISS and cycles 4.1 was used. To measure the energy consumption to study the effects of memory-boundedness of applications, Intel's Running Average Power Limit (RAPL) interface was used in combination with performance counters. RAPL is a power modeling and measuring interface which is focused on energy consumption. RAPL is an accessible and powerful tool used to create a overview of the energy characteristics of an execution.

$$Memory - boundedness = \frac{CYCLE\_ACTIVITY : STALLS\_L3\_MISS}{Cycles} \quad (4.1)$$

Table 4.2: Multi-Grid benchmark (MG), high intensity of memory-misses and accesses. Expected to increase when increasing frequency and number of threads. *Negative values and values above 100% are considered errors, since that occurrence is not possible (*).*

Thread 1	**0.8GHz**	**2GHz**	**3GHz**	**4.4GHz**
Goel-Formula	26.50%	30.30%	44.60%	54.10%
Spiliopoulos-Formula	24.30%	42.60%	49.30%	56.70%
Molka-Formula	20.60%	39.70%	46.80%	54.50%
L3-Stall formula	10.70%	19.50%	21.90%	24.90%
Threads 2	**0.8GHz**	**2GHz**	**3GHz**	**4.4GHz**
Goel-Formula	6.30%	103.60%*	56.70%	65.80%
Spiliopoulos-Formula	30.60%	40.60%	55.30%	63.30%
Molka-Formula	26.80%	31.80%	52.90%	61.20%
L3-Stall formula	17.50%	18.60%	32.80%	37.80%
Threads 4	**0.8GHz**	**2GHz**	**3GHz**	**4.4GHz**
Goel-Formula	25.40%	45.60%	98.70%	99.10%
Spiliopoulos-Formula	32.70%	51.30%	59.90%	70.10%
Molka-Formula	28.70%	43.60%	53.50%	67.40%
L3-Stall formula	19.70%	28.60%	34.30%	42.90%

The microbenchmark will have various settings of memory-boundedness. To have various settings it is important to understand what generates the highest and lowest amount of memory-boundedness to tune the rest of the settings. The lowest amount of memory-boundedness was created by looping over simple additions and multiplications of a variable. The highest amount of memory-boundedness was achieved by using a technique called pointer-chasing meaning we access the value of the first address of each cache-block with a pointer and store that value in the next cache-block. Pointer-chasing is used to generate L3-misses by knowing the size of the cache-blocks (64 bytes). The accesses are used once on each cache-block so that each instruction creates an L3 cache miss. Prefetching should be able to be turned on and misses would still occur since the instruction for the miss will be saved in the location of the value accessed in the previous block. Since only one block can be accessed at a time misses will occur.

The other settings are created by looking at values of memory-boundedness generated and then investigating which percentages were missing to cover the widest range of memory-boundedness behaviour. Each setting between lowest and highest was created by using a mixture of misses, hits and arithmetic instructions such as additions or multiplications. These cases were then aggregated to run sequentially, so one run of the microbenchmark runs all settings in order, when one would want it to train a DVFS predictor. The microbenchmark with the independent settings will still be used for research purposes or reconfiguration of settings.

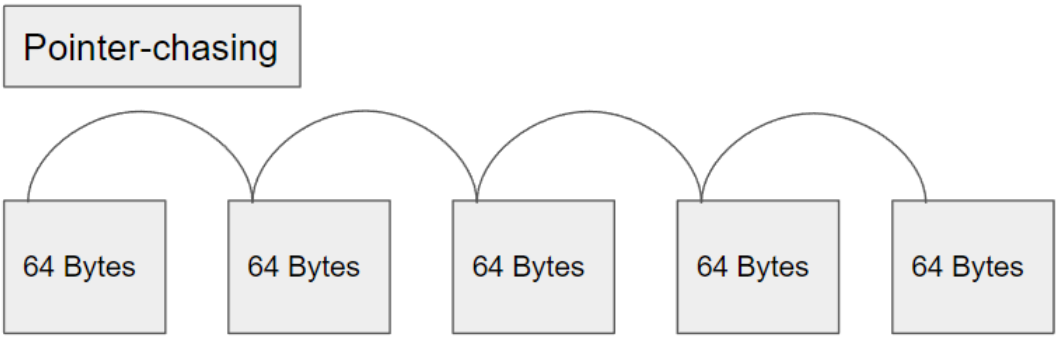


Figure 4.1: Basic illustration of the technique pointer-chasing

4.2.1 Implementation and testing the microbenchmark

The aim of the microbenchmark is to have different settings of memory-boundedness, the first goal is to create the most memory intensive program possible. To create a high level of memory-boundedness each memory access should be a miss. Misses cause stalls on the pipeline, because the execution waits for memory to return a value necessary for a continuation of the execution.

To generate misses, the size of cache blocks and last level cache needs to be known. The machine used for creating this microbenchmark has a cache block size of 64 bytes and the size of last level cache is 12 MB. To generate misses the working-set-size needs to be larger than 12 MB or else each cache block will fill the last level cache. Each cache block will be read from only once. The first address of each cache block is loaded and the rest of the cache block is ignored, so that each load instruction is a miss and causes a last level cache miss.

Another technique to increase the amount of memory-boundedness is loop-unrolling, which increases the degree of load instructions per loop so that less cycles are used as overhead to run the loop. The optimal amount of instructions per loop for high memory-boundedness was determined to be 16 load instructions, if one were to go higher the memory-boundedness would decrease. Combining these techniques and using different memory-boundedness formulas, a memory-boundedness of between $73 - 85\%$ could be achieved, depending on which formula was used.

The memory-boundedness of the microbenchmark can be configured to produce different variations. The variations of memory-boundedness is instructions that generates misses, instructions that hit and instructions which make a computation. The variations can be chosen in the setting of 0, 10, 20, 23, 25 30, 35, 40, 45, 50, 55, 60, 70 and highest possible, percentage of memory-boundedness calculated from using the formula created in this project. The equation 4.1 shows how the calculation of the memory-boundedness for each configuration is done, the microbenchmark is configured for frequency 4 Ghz, the percentage of the settings is expected to decrease if a lower frequency is used or increase if a higher frequency is used.
To add parallelism to the microbenchmark OpenMP was used. The working-set-size

scales with number of threads and each thread is working independently on its data. The pseudo-code below shows how the microbenchmark is structured and what is required to function accurately. To replicate the results, the system needs to have the same cache-block size, working-set-size, processor speed and memory-speed. The memory-boundedness is configured by using memory and computational instructions in combination. The main ways to increase or decrease memory-boundedness is by using loop-unrolling, memory misses, memory hits and computational instructions. The memory-boundedness instruction combination is ran 1000 times to ensure more accurate and reliable measurements.

In order for the memory-bound setting to be accurate the system is required to have the same cache-block size, working-set size needs to be greater than cache-block size to generate misses. CPU and RAM speed needs to be the same as the machine used to configure these settings. The instructions executed for each memory-boundedness will remain but can be adjusted for other system settings. Algorithm 1 shows the code of the microbenchmark which run one setting depending on user input. Algorithm 2 shows the code of the microbenchmark when running all settings in order. For testing and tuning of the settings Algorithm 1 was used. Algorithm 2 is the final microbenchmark which has a wide range of coverage of memory-boundedness behaviours in one execution. The results of running the microbenchmark will vary depending on which platform it is ran on, but will still cover a wide range of memory-boundedness even if results differ. The setting for 23% was used to fill large gaps discovered in the results when increasing number of threads, to have a larger variety of memory-boundedness.

Algorithm 1 Code for testing each setting

Require: Cache-block size = 64 bytes
Require: Cache size < Working-set-size
  if $INPUT$ is A **then**
    for $i = 0; i < 1000; i++$; **do** 70% Memory-boundedness (Highest amount of Memory-boundedness)
    end for
  else if $INPUT$ is B **then**
    for $i = 0; i < 1000; i++$; **do** 60% Memory-boundedness
    end for
  else if $INPUT$ is C **then**
    for $i = 0; i < 1000; i++$; **do** 55% Memory-boundedness
    end for
  else if $INPUT$ is D **then**
    for $i = 0; i < 1000; i++$; **do** 50% Memory-boundedness
    end for
  else if $INPUT$ is E **then**
    for $i = 0; i < 1000; i++$; **do** 45% Memory-boundedness
    end for
  else if $INPUT$ is F **then**
    for $i = 0; i < 1000; i++$; **do** 40% Memory-boundedness
    end for
  else if $INPUT$ is G **then**
    for $i = 0; i < 1000; i++$; **do** 35% Memory-boundedness
    end for
  else if $INPUT$ is H **then**
    for $i = 0; i < 1000; i++$; **do** 30% Memory-boundedness
    end for
  else if $INPUT$ is I **then**
    for $i = 0; i < 1000; i++$; **do** 25% Memory-boundedness
    end for
  else if $INPUT$ is J **then**
    for $i = 0; i < 1000; i++$; **do** 23% Memory-boundedness
    end for
  else if $INPUT$ is K **then**
    for $i = 0; i < 1000; i++$; **do** 20% Memory-boundedness
    end for
  else if $INPUT$ is L **then**
    for $i = 0; i < 1000; i++$; **do** 10% Memory-boundedness
    end for
  else if $INPUT$ is M **then**
    for $i = 0; i < 1000; i++$; **do** 0% Memory-boundedness (No Memory-boundedness)
    end for
  end if

Algorithm 2 Code of Microbenchmark

Require: Cache-block size = 64 bytes
Require: Cache size < Working-set-size
for $i = 0; i < 1000; i++$; **do** 70% Memory-boundedness (Highest amount of Memory-boundedness)
end for
for $i = 0; i < 1000; i++$; **do** 60% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 55% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 50% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 45% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 40% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 35% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 30% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 25% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 23% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 20% Memory-boundedness
end for
for $i = 0; i < 1000; i++$; **do** 10% Memory-boundedness
end for
for $i = 0; i < 1000; i{+}{+}$; **do** 0% Memory-boundedness (No Memory-boundedness)
end for

5
Results

The results of this thesis will present the NAS benchmark suite memory-boundedness coverage and the coverage of the microbenchmark. Then a comparison between them is done for memory-boundedness, energy and execution time.

5.1 NAS benchmark suite

NAS benchmark suite has a large variety of benchmarks which covers many different features of memory-boundedness and other application features. The focus of the graphs 5.1 5.2 5.3 will be on memory-boundedness and the graphs will be presented in order of how many threads the benchmarks were run on. When using multiple threads, each thread was allocated to a different core and CPU affinity was used for all cases. The formula 4.1 was used to calculate memory-boundedness on all graphs. Other formulas were only used in the testing and investigation process, but would present similar memory-boundedness patterns but not be as accurate and simple to measure. The graphs shows the data points for runs on 0.8 GHz, 2.6 GHz and 4.4 GHz. The lines between data points are only drawn to show a correlation between points and does not show the actual memory-boundedness for all frequencies, even if that could possibly be the case.

The selected NAS parallel benchmarks will run with varying working-set-sizes A, B, C and W for the graphs presented.

- IS - Integer Sort, random memory access
- EP - Embarrassingly Parallel
- CG - Conjugate Gradient, irregular memory access and communication
- MG - Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive
- FT - discrete 3D fast Fourier Transform, all-to-all communication
- SP - Scalar Penta-diagonal solver
- LU - Lower-Upper Gauss-Seidel solver
- UA - Unstructured Adaptive mesh, dynamic and irregular memory access

Observation 1 *The view of the theoretically explained memory-boundedness in Figure 2.1 strengthens, since the actual memory-boundedness behaves similar to what is expected when changing frequency.*

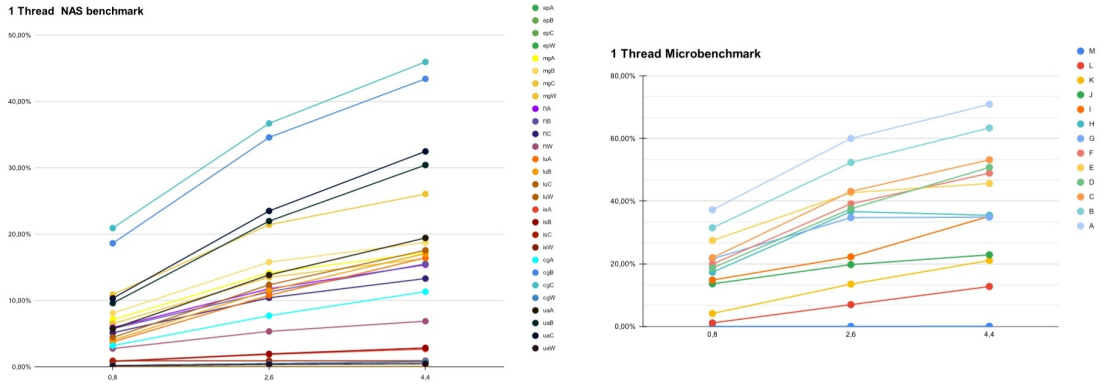


Figure 5.1: Graph of memory-boundedness coverage for frequencies 0.8 GHz, 2.6 GHz and 4.4 GHz on 1 thread from the benchmarks of the NAS benchmark suite and the Microbenchmark. Y-axis is memory-boundedness and X-axis is the CPU frequency.

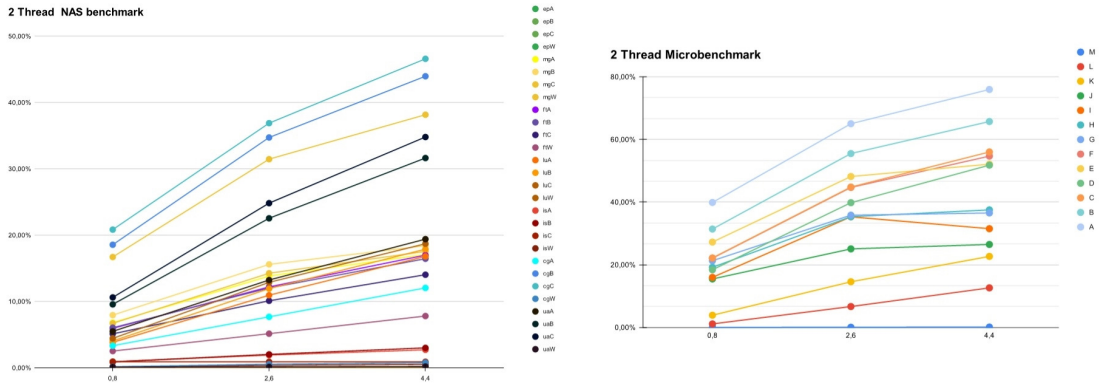


Figure 5.2: Graph of memory-boundedness coverage for frequencies 0.8 GHz, 2.6 GHz and 4.4 GHz on 2 threads from the benchmarks of the NAS benchmark suite and the Microbenchmark. Y-axis is memory-boundedness and X-axis is the CPU frequency.

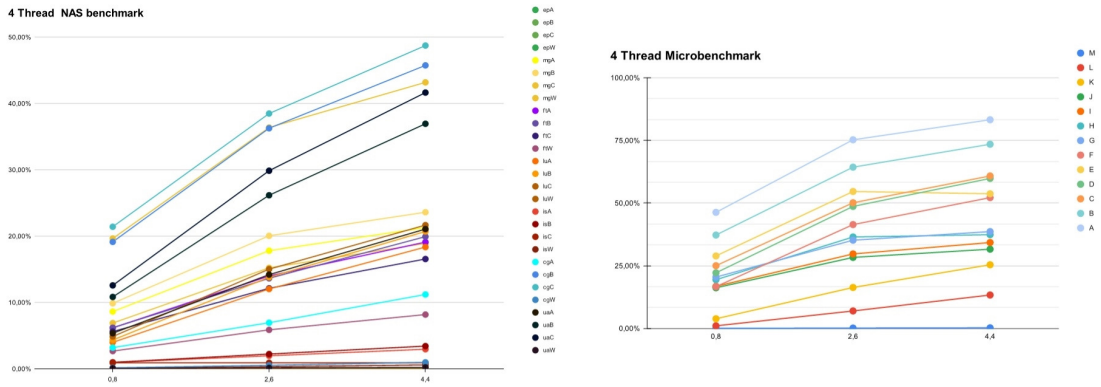


Figure 5.3: Graph of memory-boundedness coverage for frequencies 0.8 GHz, 2.6 GHz and 4.4 GHz on 4 threads from the benchmarks of the NAS benchmark suite and the Microbenchmark. Y-axis is memory-boundedness and X-axis is the CPU frequency.

Observation 2 *The formula 4.1 is a valid estimate for calculating similar values as what is theoretically expected. Since it shows expected behaviour when increasing CPU frequency.*

Observation 3 *As more threads and cores are used, a more complex correlation between memory-boundedness and number of threads can be seen in the data. Especially in the case of MG working-set-size C.*

5.2 Microbenchmark

The microbenchmark has a variety of settings which covers several different intensities of memory-boundedness. The focus of the graphs 5.1, 5.2 and 5.3 will be on memory-boundedness and the graphs will be presented in order of how many threads the benchmarks were run on. When using multiple threads, each thread was allocated to a different core and CPU affinity was used for all cases. The formula 4.1 was used on all graphs. Other formulas were only used in the testing and investigation process. The graphs show the data points for runs on 0.8 GHz, 2.6 GHz and 4.4 GHz. The lines between data points are only drawn to show a correlation between points and does not show the actual memory-boundedness for all frequencies, even if that could possibly be the case.

Observation 4 *The microbenchmark covers a wide range of memory-boundedness behaviour between 0 and 83%.*

Observation 5 *Some settings of the microbenchmark generates unpredictable memory-boundedness behaviour, contradicting the assumption that increasing frequency and number of threads always increases memory-boundedness. Could be caused by overlapping of memory-misses at higher CPU frequencies, but no definite explanation*

has been made.

5.3 Comparison of NAS and Microbenchmark

The comparison shows the values of memory-boundedness, energy and execution time by NAS and the microbenchmark (MICRO). The NAS benchmark suite has more data points of settings but produces a lesser range of memory-boundedness than the microbenchmark in all cases.

5.3.1 Memory-boundedness

When comparing the data set of memory-boundedness for the microbenchmark with the NAS on one thread, the microbenchmark shows a wider range. The NAS shows more consistent and similar values which does not cover as a wide range of memory-boundedness. The microbenchmark covers from 0 to 37% on 0.8GHz compared to the NAS which covers from 0 to 21%, as can be seen in figure 5.1. When increasing the frequency both of them however shows an increase in memory-boundedness for all settings and at the highest frequency the microbenchmark covers from 0 to 71% and the NAS 0 to 47%, as can be seen in figure 5.1. The graph of two threads 5.2 show similar correlation between frequency and memory-boundedness to the single threaded graph 5.1. Since the difference is in number of threads and it can be seen that for most points of the graph the memory-boundedness is higher when using more threads. The four threaded graph 5.3 strengthens this assumption and also show the same correlations with frequency and memory-boundedness.

Observation 6 *When using the formula 4.1 the microbenchmark covers a wider range of memory-boundedness behaviours than the selected benchmarks from the NAS benchmark suite. The microbenchmark covers a wider range for all settings of frequency and number of threads.*

5.3.2 Energy and Execution Time

The results in figures 5.4, 5.5 and 5.6 show the comparison of trends in energy consumption from a NAS benchmark which exhibits similar memory-boundedness behaviour to the setting compared with from the microbenchmark. The figures show energy and execution time at different frequencies and ideally the NAS benchmark should show similar behaviour as the microbenchmark with similar memory-boundedness behaviour. The benchmarks EP, MG and CG from the NAS benchmark suite are compared with their memory-boundedness counter-parts from the microbenchmark settings 0%, 25% and 45%.

Observation 7 *In Figure 5.4 the energy consumption and execution time between EP and 0% setting of microbenchmark have very similar patterns. With the excep-*

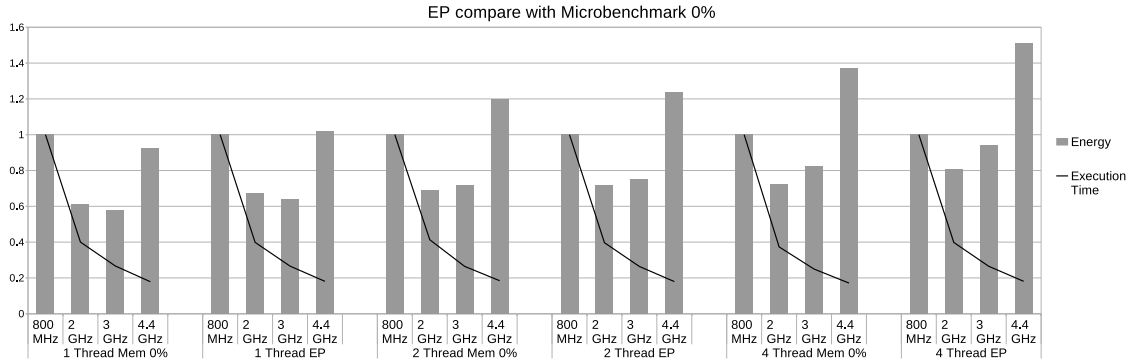


Figure 5.4: Graph of normalized energy usage and normalized execution time for frequencies 0.8 GHz, 2 GHz, 3 GHz and 4.4 GHz on NAS benchmark EP and setting 0% memory-boundedness on the microbenchmark.

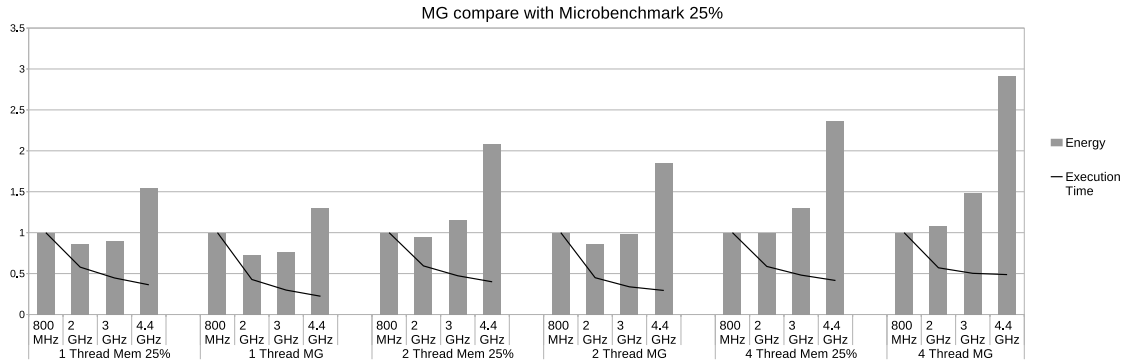


Figure 5.5: Graph of normalized energy usage and normalized execution time for frequencies 0.8 GHz, 2 GHz, 3 GHz and 4.4 GHz on NAS benchmark MG and setting 25% memory-boundedness on the microbenchmark.

tion of EP is slightly more energy consuming.

Observation 8 *In Figure 5.5 the energy consumption and execution time between MG and 25% setting of microbenchmark vary in both energy and execution time scaling. The MG benchmark scales significantly better than the 25% setting when increasing frequency, except on four threads.*

Observation 9 *In Figure 5.6 CG and 45% setting of microbenchmark vary the most of the selected cases in both energy and execution time scaling. The CG benchmark scales significantly better than the 45% setting when increasing frequency. The 45% setting seems to be too memory-bound to be a accurate counter-part for the CG benchmark.*

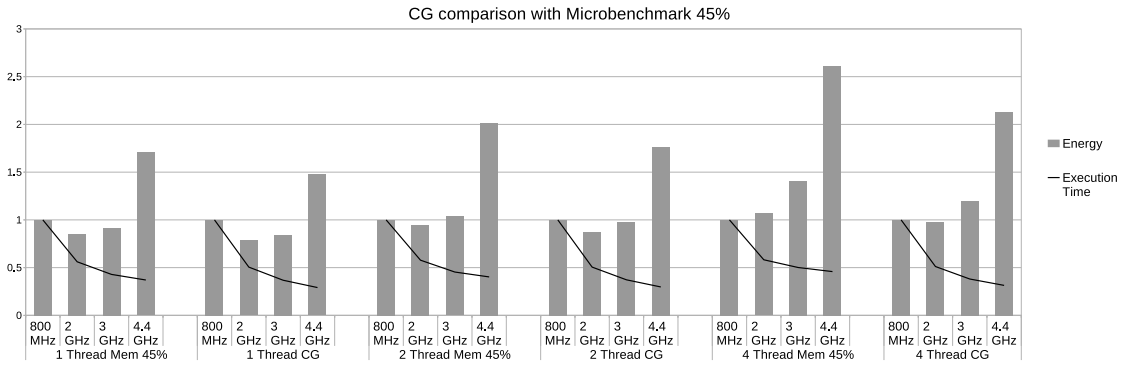


Figure 5.6: Graph of normalized energy usage and normalized execution time for frequencies 0.8 GHz, 2 GHz, 3 GHz and 4.4 GHz on NAS benchmark CG and setting 45% memory-boundedness on the microbenchmark.

6
Conclusion

The conclusion of this thesis will be covering the method, the results and the future of the project. The conclusion covers what could have been improved further and potential errors to consider and should be taken into account.

6.1 Method

The method used to measure memory-boundedness was the formula 4.1, which was created to be a simple and effective method of comparing settings and benchmarks. The formula 4.1 worked as intended and the result from it produced similar memory-boundedness patterns as the other formulas 3.1, 3.2 and 3.3. The formula 4.1 was simpler, easier to calculate in all cases and correlated to the assumptions made about memory-boundedness. One weakness with the formulas 3.3, 3.2 and 4.1 is that they are based on Intel specific performance counters, so one would have to look for other similar performance counters on their processor to recreate the usage of formulas.

6.2 Results

The result which was achieved suggests that the microbenchmark covers a wider range of memory-boundedness than the NAS benchmarks. A conclusion which can be made is that for simulating a specific system characteristic, creating a microbenchmark to simulate the wanted characteristic is the superior approach in the case of memory-boundedness. However since the range of memory-boundedness on the microbenchmark has not been used to train a DVFS predictor, the usefulness of the microbenchmark cannot be determined or measured in accuracy of predictions compared to the NAS benchmarks.

The expected results of memory-boundedness was assumed theoretically and described in the Figure 2.1 for the formula 4.1. The results suggest that this assumption is accurate, however the correlation between number of threads and memory-boundedness is a feature which should be investigated further. The increasing of number of threads each on different cores led to increase in memory-boundedness most commonly, but in the CG benchmark the change was significantly less. An explanation for this behaviour could be that communication between cores leads to larger memory-stalls and the execution therefore becomes more memory-bound.

The expected energy and execution time of the NAS and the corresponding setting on the microbenchmark, was expected to be similar. For the EP benchmark the microbenchmark had the same pattern in execution time but slightly lower energy consumption. The amount of time EP runs is significantly higher than the microbenchmark on setting 0%. This could lead to voltage leakage by higher temperature of the CPU, explaining the higher energy consumption. The MG benchmark and the microbenchmark setting 25% produce different patterns when compared. The MG benchmark scales execution time better than what was expected running on 1 or 2 threads, but on 4 threads the MG benchmark scales worse and consumed more energy than anticipated. This behaviour suggests that the MG benchmark has an unexpected increased amount of memory-boundedness from the communication between the 4 threads compared to the microbenchmark setting 25%. To the contrary of the MG benchmark, CG seems to scale more efficiently with CPU frequency having almost the same amount of memory-boundedness when increasing number of threads. Which would suggest that the threads do not communicate to the same degree as MG or the corresponding microbenchmark setting.

Creating a microbenchmark specified to create large variations of memory-boundedness behaviour was a success. But the degree of usefulness and importance the microbenchmark has on creating more accurate prediction models cannot be validated, since no tests and measurements has been done. However the microbenchmark has led to powerful insights in how the memory-boundedness differentiates depending on the amount of threads used. The threads communicate mostly on L1- and L2 cache. The formula 4.1 does not calculate these misses, the memory-boundedness when increasing number of threads can be inaccurate. If one were to create a new formula for memory-boundedness, the communication stalls between threads should be considered.

6.3 Future work

The next step would be to test the microbenchmark by training a DVFS predictor with it and comparing the prediction accuracy with a DVFS predictor trained from the benchmarks of NAS. After evaluating the results, a comparison of the prediction models would be investigated, then modifications and settings could be added to the microbenchmark. Features that would have been interesting to add already would be settings for different memory-boundedness between threads and creating an easier path for creation of new memory-boundedness settings. It would also be interesting to look at more data from adding more variations of frequencies. This would lead to adding more memory-boundedness settings to the microbenchmark, which could improve the prediction models further by having a larger data set.

Overall the microbenchmark can be used to train a DVFS predictor and does cover a larger variety of memory-boundedness behaviour based on performance counters compared to NAS benchmarks. Understanding how number of threads affect memory-boundedness would lead to more accurate settings for multi-threading. Adding settings for memory-boundedness behaviour, which scales with number of

threads and no scaling would improve the microbenchmark further, covering more cases of memory behaviours.

References

[1] Eyerman, S., and Eeckhout, L. A counter architecture for online dvfs profitability estimation. *IEEE Transactions on Computers 59*, 11 (2010), 1576–1583.

[2] Goel, B. *Measurement, Modeling, and Characterization for Energy-efficient Computing*. Chalmers University of Technology and Göteborg University, 2016.

[3] Guo, Z., Bhuiyan, A., Liu, D., Khan, A., Saifullah, A., and Guan, N. Energy-efficient real-time scheduling of dags on clustered multi-core platforms. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2019), IEEE, pp. 156–168.

[4] Jin, H., Frumkin, M., and Yan, J. The openmp implementation of nas parallel benchmarks and its performance. Tech. rep., Citeseer, 1999.

[5] Keramidas, G., Spiliopoulos, V., and Kaxiras, S. Interval-based models for run-time dvfs orchestration in superscalar processors. In *CF '10* (2010), pp. 287—-296.

[6] Maghsoud, Z., Noori, H., and Mozaffari, S. P. Peps: predictive energy-efficient parallel scheduler for multi-core processors. *The Journal of Supercomputing* (2021), 1–20.

[7] Miftakhutdinov, R., Ebrahimi, E., and Patt, Y. N. Predicting performance impact of dvfs for realistic memory systems. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture* (2012), pp. 155–165.

[8] Molka, D., Schöne, R., Hackenberg, D., and Nagel, W. E. Detecting memory-boundedness with hardware performance counters. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering* (New York, NY, USA, 2017), ICPE '17, Association for Computing Machinery, p. 27–38.

[9] Spiliopoulos, V., Kaxiras, S., and Keramidas, G. Green governors: A framework for continuously adaptive dvfs. In *2011 International Green Computing Conference and Workshops* (2011), IEEE, pp. 1–8.

[10] Su, B., Greathouse, J. L., Gu, J., Boyer, M., Shen, L., and Wang, Z. Implementing a leading loads performance predictor on commodity processors. In *2014 {USENIX} Annual Technical Conference ({USENIX} {ATC} 14)* (2014).

[11] Tzilis, S., Trancoso, P., and Sourdis, I. Energy-efficient runtime management of heterogeneous multicores using online projection. *ACM Transactions on Architecture and Code Optimization (TACO) 15*, 4 (2019), 1–26.

A

Appendix 1