The Cost of Computation: Metrics and Models for Modern Multicore-based Systems in Scientific Computing

Master Thesis in Computational Engineering

Ayesha Afzal
born 11.11.1991
Matriculation no. 21800113

Advisor
Prof. Dr. G. Wellein
Dr.-Ing. G. Hager

INSTITUT FÜR INFORMATIK
Regionales Rechenzentrum Erlangen (RRZE)
FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG

Declaration:

I confirm that I developed this thesis on my own, without any help of others and that no sources and facilities other than those mentioned in this thesis were used. This thesis has never been submitted in total, in part or in modified form to any other examination board. All quotations taken from other sources are referenced accordingly.

Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß ubernommen wurden, sind als solche gekennzeichnet.

Ayesha Afzal
Erlangen July, 2015 .....

Acknowledgments

I would like to take this opportunity to express my profound gratitude to my parents in Pakistan and my husband for their continuous support and encouragement during the course of master's studies at FAU Erlangen. Working in the productive research environment of the High Performance Computing group was a learning experience and I am thankful to Prof. Dr. Gerhard Wellein for providing me an opportunity to do my thesis work at the institute. This work provided me with an invaluable insight into many aspects of modern multi-core-based systems.

I want to pay my special thanks to my thesis advisor Dr. Georg Hager for his valuable guidance, kind behaviour and help in a light and friendly environment throughout the entire duration of this work. His support and interesting discussions were always refreshing and especially helpful in evaluating the results and writing this thesis. I am grateful to him for spending time read this thesis and providing useful suggestions about this thesis. Furthermore, I wish to express my sincere thanks to Moritz Kreutzer for providing the "27-point stencil matrices" for evaluating my conjugate gradient solver and Holger Stengel for helping me with "Jacobi" code.

Lastly, I would like to acknowledge Thomas Röhl, Dr. Thomas Zeiser, Faisal Shahzad and my other colleagues for their help and support in understanding the world of HPC. The initial days of working with available multi-core-based systems and tools were made easy by their useful discussions.

Table of Contents

Chapter 1

Abstract

The increasing concern on power consumption in many computing systems points to need for power/energy modelling and estimation of high-end computing systems. The goal of the present work is to propose power/energy models that can predict the run-time energy consumption of loop kernels and programs by specifying their properties with respect to scaling behaviour, data transfer through the memory hierarchy, and low-level operations. This model should ultimately be able to answer following questions:
"How algorithm properties may help to inform power management?"
"How can a program be run so that the overall energy consumption is minimized without compromising time to solution?"
"How can a program can be executed so that the overall energy consumption is minimized, with a maximum increase in time to solution?"
"How to treat a problem slowdown in case of the power capping? What is the potential of larger machines towards energy saving by compensating for the slow code?"

This work proposes component level (CPUs and DRAMs) power and/or energy models and synthesizes a significant number of techniques to get an energy-efficient system for a wide range of situations by focusing on common architectural characteristics. To validate predicted models and results, experimental measurements based on the selective benchmarks are compared against estimated model's outcomes on multi-core based systems.

The present work describes the characteristics of different multi-core processors available in RRZE computing centre by elaborating every micro-architecture change with a die shrink of the process technology. A comparison of these systems is performed from the perspective of the performance, the power dissipation and especially relevant from considerations of the energy consumption.

A statistical analysis of power characteristics is obtained to get an idea about variations of the power dissipation across the Ivy Bridge EP processor in "emmy" production system available at RRZE centre. This enables us to define a policy for power aware scheduling on "emmy" cluster and thereby saves energy cost of "emmy" cluster.

This work also elaborates how different type of processor executed instructions and the tunable intensity benchmarks effect the power dissipation and its model parameters W_i on recent processors. In addition, it elaborates that low level microscopic parameters (i.e., energy cost for one flop or for one byte transfer) between memory hierarchy levels predict energy cost at macroscopic level. Finally these two modelling techniques are combined to determine macroscopic W_i parameters form these low level microscopic parameters.

Chapter 2

Introduction

The huge power consumption of modern high performance microprocessors with increasing clock speed and circuit density is becoming increasingly important in design consideration, not only in mobile processors, servers and data-centres, but also in high-performance supercomputers. Today, each market segment has its own power requirements and constraints, making power limitation an imperative factor in any new micro-architecture. This means that some segments are designed to achieve the highest performance possible while other systems are optimized for best performance under a given power cap.

In the past years, microprocessors have gone through significant changes. Each new microprocessor generation introduces higher frequency that increases work per instruction and results in a larger total power consumption of the microprocessor. Improvements in fabrication process technology and microprocessor micro-architecture have resulted in the phenomenon known as "Moore's law" [1] which stated that the chip density doubles every 18 to 24 months. With increase in core counts on processors, the high-end computing systems built with multi-core processors provide thread-level parallelism. Programmers tend to use OpenMP [2] or message passing interface MPI [3] programming models, to exploit multiple cores on a cache-coherent shared-memory node or on clusters of nodes to obtain efficient execution of parallel applications, respectively. Thus, steadily growing transistor budget, which has produced many more transistors on a chip running at much higher frequencies, has drastically raised the power dissipation. Today, it has become an issue in high performance microprocessors.

When adding complexity to exploit performance, one must understand and consider the power impact. The arrival of multi-threaded and multi-core architectures exaggerates thermal problems, since concentrating more computational activity on smaller chip areas leads to increased chip temperature. For high performance computing, clock frequencies continue to increase, causing the dissipated power to reach the thresholds of current packaging technology. If the trend persists, soon we may experience a chip with the power density of a nuclear power plant. Consequently, the improved cooling solutions and the power distribution are utilized for reducing high power consumption. However, such components are costly and cannot be expected to scale to higher power levels as transistor dimensions shrink.

So far only the active power has been primarily considered whereas the leakage power is ignored. The leakage power is dissipated by current running in gates and wires even when there is no activity. Similar to the active power dissipation which increases as a result of increasing performance and complexity, the enhanced static power in terms of sub-threshold drain-source leakage is because of lower threshold voltage usage to gain performance; the higher gate-substrate

leakage power is reflected by the oxide thickness scaling with technology scaling and the increased junction leakage power is justified by higher power densities. As such, controlling power density has become a major challenge and even more crucial in future technologies.

For a long time, the only metric that was used for assessing supercomputer performance was their "speed" (see Top500 list ranking [4]). However, nowadays the power dissipation and power density have become a critical design constraint by limiting system performance. Therefore, the architecture which maximizes the "performance per watt" (power efficiency) will maximizes performance (see Green500 list ranking [5]).

2.1. Fundamental metrics for characterising a multi-core processor

During the previous years, microprocessors have gone through significant variations; however, the basic computational model operating on the architecture hierarchical states (i.e., memory and registers) remains same. A program consists of data and instructions encoded in a specific instruction set architecture, ISA. To understand the trade-offs of various designs, following different key metrics are considered in present work [6], [7].

Microprocessor complexity is primarily determined by the effort it requires to manufacture a microprocessor. This metric is discussed only implicitly in present work as it is reflected by other microprocessor metrics such as performance, power and cost metrics. Thus, the enhanced complexity of the microprocessor causes an increase in its energy consumption in processing an instruction.

Microprocessor performance reflects the time it takes to complete a task, since time to solution is inverse performance with an assumption that we always run the same problem. Performance is affected by the specific workload, compiler optimizations, instruction set architecture, operating architectural configuration, and many more. A "Cycle Per Instructions (CPI)" metric can be observed from a performance perspective, since an enhancement in performance can be achieved by increasing the number of clock cycles per second or by decreasing the average number of cycle per instructions, CPI.

Microprocessor power shows energy performance product and is measured in watts. However, two different metrics are utilized to characterize the power of modern processors i.e, "CPU Power" metric and "DRAM Power" metric. These metrics measure the power consumed by the processor and the power dissipated by the memory controller when running a particular application. Higher performance demands more power. However, the power dissipation is constrained due to thermal dissipation and power density (i.e., power dissipated by the chip per unit area, in watts/cm^2), and thus causes a limit in performance growth. In order to keep transistors within their temperature limits, the heat generated due to increased power density need to be dissipated from the source in a cost-effective manner. However, the latest x86 systems already contain support for multiple sleep states with various latencies and standby power levels. A lot of research has been dedicated to exploiting the benefits of these extra states by doing energy management of the inactive components, which implies minimum total energy consumption of sleeping machines [8].

Microprocessor cost is measured in terms of the physical size of the manufactured silicon die. Larger die area causes higher power consumption, thus the increased manufacturing cost.

Microprocessor energy efficiency is another important power and/or performance related metric which reflects the power to performance ratio, in Joules. Event monitoring counters included in modern microprocessors can be used to create energy profiles describing the energy

characteristics of individual tasks. In order to gauge operating efficiency, "Energy to solution" metrics for both CPU and DRAM are utilized, since total energy per unit of work is concerned. An alternative is to employ the "Energy Delay Product (EDP)" and its generalization metrics as they provides more information than a pure energy to solution metric.

2.2. Role of power consumption in supercomputing

Modern multi-core chips show complex behaviour with respect to performance and power. Besides performance aspects, considerations on power dissipation of multi-core chips have become more popular in supercomputing. It is highly conceivable that in future, processor designs will not only be faster but also more power-efficient than their predecessors.

In high performance computing centres, each cluster is designed to support a certain kilowatts hours (kWh). This means that the higher power causes shorter operating time of a cluster. Lets consider a cluster of a dual socket nodes in which every node includes CPU, DRAM, network, disk, and many more components. Each cluster node dissipates a maximum power of 150 Watts per socket for both central processing unit, CPU, and installed random access memory, RAM. However, the baseline power of rest of system, which includes disks, network, and more, is about 100 Watts which implies that the system node has overall power dissipation of 400 Watts per node.

Such a cluster node roughly cost around 4000 € and consumes 400 Watts of power within six years of its operational runtime. The consumed kilowatts hours for this node can be mow calculated with price of electric power.

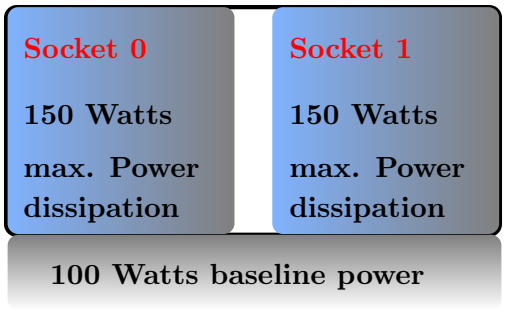


Figure 2.1.: Energy-cost model for a dual-socket production cluster node

Electricity prices vary between countries and can even vary within a single region or distribution network of the same country. For example, the energy cost in Germany is 0.298 €/kWh for private households and 0.16 €/kWh for the university whereas it costs 0.06 €/kWh in USA. Moreover, Power Utility Efficiency (PUE) factor helps to estimate the amount of power spend over six years of operational runtime of this node by considering the additional cost of dedicated cooling, beyond its actual electric power dissipation. At RRZE centre, PUE is about 2.5, i.e., for every Watt of electric power we have to spend about 1.5 Watts in addition for cooling while at LRZ centre, PUE is of 1.2, i.e., they only have 20% overhead for cooling, since they use free hot water cooling for most of the year.

These numbers gives a rough estimation of energy cost depending on the location where this node is running. For instance, the energy cost at RRZE or LRZ computing centre is 8415 € or 4039 € in addition to the hardware cost of 4000 €, respectively, which is beyond the budget constraint of some high-end computing centers. In the power-constrained systems, such

information is useful for programmers to explore the design space for energy-efficient system and to assure that overall system power saves a non-negligible amount of money by increasing the "science per €" to meet the specified budget.

2.3. Understanding Power behaviour using analytical models

The increasing concern on power issues in many computing systems points out the need for the power modelling and estimation for high-end computing systems. Therefore, accounting power is imperative for accurate power modelling in the light of intensive workloads. Moreover, the power estimation can help to estimate how optimization effects the complete system behaviour from the perspective of energy consideration.

The most striking observation is the strong correlation between power consumption and the microprocessor performance during application execution. As tools and the methodology to measure these metrics exist on modern microprocessors, analytical models can be proposed to estimate the power dissipation for run time energy management. The detail and explanation of these proposed models will come in Chapter 5.

2.4. Previous work

A refined performance modelling of streaming loop kernels has been done through "ECM model" [9], [10], to extend a "roofline model" [11] and to predict a more accurate description of the serial and parallel performance of codes by developing interaction of code with the hardware. Hager et al.[10] constructed an elementary CPU power model from some derived simplified assumptions by exploring the power dissipation characteristics of the Sandy Bridge processor on several benchmark codes. They suggested that the dynamic CPU power dissipation is a second-degree polynomial,

$$W(f,n) = \quad W_0 + W_1 nf + W_2 nf^2 = \quad W_0 + w_1 f + w_2 f^2 \tag{2.1}$$

with $w_{1,2} = W_{1,2}$ n and it implies the following conclusions

- The dynamic power dissipation W(f, n) is a quadratic polynomial in the clock frequency with deviations from some baseline frequency f_0 such that $f = (1+\Delta\nu)f_0$, where $\Delta\nu = \frac{\Delta f}{f_0}$, which is parametrize by w_2. This quadratic part w_2 depends strongly on the characteristics of running application, and has some inverse relation with the performance metric "CPI".
- The linear factor w_1 is generally small compared to w_2.
- The baseline or leakage power of the chip W_0 with powered on, is independent of the type of application executed, the number of active cores and the clock speed. However, it is different from the reported "idle power" of the chip, which is considerably lower due to advanced power gating mechanisms.
- The dynamic power dissipation W(f, n) has a linear dependence on the number of active cores, n, in the non-saturated regime. However, the type of running application has much less influence on power dissipation than the number of active cores, which is reflected by a slow increase in the power dissipation per core in the saturation regime compare to non-saturated regime.

- Although hyper-threading result in the performance gain depending on the running code; but at the same time, it implies high Power dissipation due to the improved utilization of the pipelines, so it may be more power-efficient to ignore the SMT threads.

Using these conclusions, Hager et al.[10] established a simple qualitative chip "energy to solution" model

$$E = \frac{W(f,n)}{P(f,n)} = \frac{W_0 + W_1 nf + W_2 nf^2}{min((1+\Delta\nu)nP_0, P_{max})} \tag{2.2}$$

to obtain optimized energy to solution of parallel codes on a multi-core chip. The performance scales linearly with the number of cores, n, and the normalized clock speed, $1+\Delta\nu$ until it hits a memory bandwidth bottleneck. This model provides following useful guidelines for energy-efficient executions of parallel applications.

- The more cores are used, the smaller energy to solution for scalable codes, whereas, for codes that show performance saturation at some $n_s = \frac{P_{max}}{(1+\Delta\nu)P_0}$, energy to solution is minimal at this point and using more cores is just wastage of energy.
- In non-saturated regime, optimal frequency for energy-efficient execution $f_{opt} = \sqrt{\frac{W_0}{W_2 n}}$ depends on the number of cores used and the ratio of baseline and dynamic power i.e., a large dynamic power factor W_2 leads to lower f_{opt}. When f $< f_{opt}$, large baseline power W_0 causes "clock race to idle" rule [12], i.e., running a processor with a high clock frequency to complete a computation as fast as possible and go to sleep as early as possible to eventually save energy. Whereas, beyond saturation point in saturated regime, minimum energy to solution is achieved at the lowest possible clock speed as it grows with the frequency.

2.5. Related work

A lot of research emphasised on reducing energy consumption through dynamic frequency and voltage scaling, DVFS, by scaling down the clock speed together with core voltage to save power without limiting performance [8], [13]. Moreover, Dynamic Concurrency Throttling (DCT), controls the number of active threads that execute shared memory applications to provide energy savings by tuning performance and power consumption. Since due to system bottlenecks (e.g., memory bandwidth), the scalability of the regions can vary significantly so DCT often simultaneously reduces both execution time and power consumption [1].

Choi and Vuduc [14] has modified the research of Demmel et al.[15] and postulate a "power line model" for power and an energy-based analogue of the time-based "roofline model" [11] that connect the properties of an algorithm with their costs in time and energy. They also defined "energy balance" analogue of "time balance" as ratio of useful compute operations to bytes per unit energy and "time-energy balance gap" as a difference between time-balance and energy-balance.

Choi and vuduc suggested that the "energy-based roofline" model [14] is actually a smooth "arch line" unlike the "time-based roofline" model [11], since energy cannot be overlapped while time can. With "time-energy balance gap", there are distinct intensity points for being "compute bound" or "memory-bound" depending on whether the aim is to save time or energy. When energy-balance overtakes time-balance, algorithmic energy-efficiency is more difficult to achieve. Consequently, energy-efficiency implies time-efficiency while the converse is not true. Today, time-balance exceeds energy-balance due to idle power and other micro-architectural

inefficiencies and causes race-to-halt [12] strategies as most reasonable technique to save energy. Their experiments show that the hypothetical "time-energy balance gap" does not yet really exist, which consequently explains why on today's platforms "race-to-halt" is likely to work well.

Their work also suggested that although the cost ratio between reading data from memory and computing on data is expected to remain constant [16], wire capacitance will not scale. Consequently, the cost of moving data will stay the same. Unless the distance between memory and the cores decreases significantly via memory-die stacking, the time-energy balance gap will increase.

2.6. Thesis organization

The rest of work is structured as follows. Chapter 3 discusses the important features and characteristics of multi core modern processors on several test systems and tools available at RRZE computing centre, which are subsequently utilized in Chapter 6 to provide a power and/or energy aware comparison of these systems. Chapter 4 analyses the performance, the power dissipation and the energy consumption characteristics under a verity of workload scenarios, that is dense matrix-matrix multiplication, jacobi stencil solver and conjugate gradient method on a multi-threaded, multi-core Sandy Bridge EP processor.

Chapter 5 presents a modification of the analytical chip power model by Hager et al. [10] and construction of an elementary DRAM power model to provide a complete model for the power dissipation and thus the energy consumption estimation. It also validates the presented models against the measurement results. Chapter 6 provides possible techniques to achieve an energy-efficient execution of scientific computing workload in best possible way for a wide range of situations. Chapter 7 presents a statistical analysis of power dissipation of Ivy Bridge EP processor for both compute and accelerator nodes in "emmy" cluster and chapter 8 provides a connection of these macroscopic model parameters W_i to microscopic level parameters (energy cost for one flop and one byte transfer). Chapter 9 concludes the results.

Chapter 3

Test Systems and Tools

This section elaborates the description and the characteristics of different test clusters and tools available at RRZE high performance computing centre to study their power consumption and the performance behaviour under a wide range of workloads. The command "Likwid-topology -g" delivers the graphical output of machines topology. Intel introduced the "Running Average Power Limit (RAPL)" energy sensors with the Sandy Brigde micro-architecture for measuring energy consumption of short code paths and now it is available in almost all recent Intel CPUs [17]. The Intel "Tick/Tock" model [18] discusses every micro-architectural change with a die shrink of the process technology as shown in Figure 3.1.

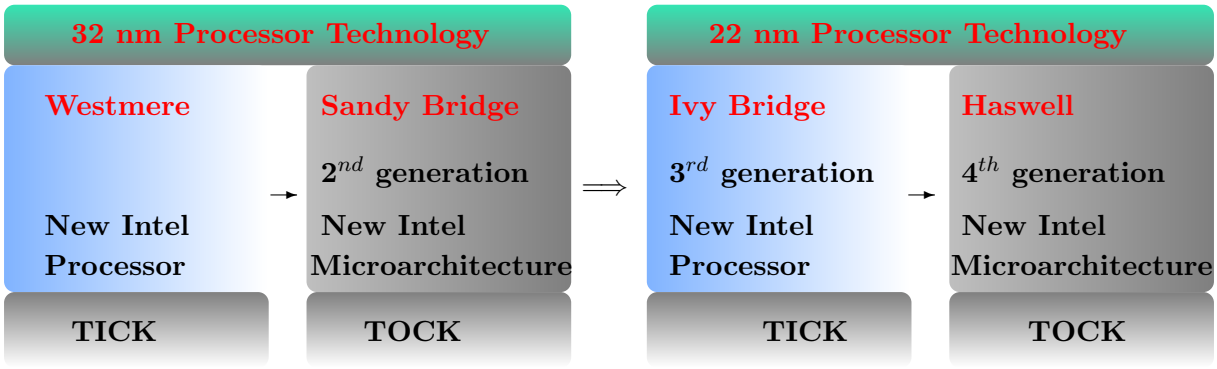


Figure 3.1.: Intel tick tock model towards Intel's next generations

3.1. "phinally" Testsystem

The "phinally" test system has a dual-socket 8 cores Intel Sandy Bridge EP processor with 16 logical cores per socket through hyper-threading (see Fig. 3.2). It operates at 2.7 GHz base clock speed and features Intel turbo mode for increased performance on an as-needed basis. Sandy Bridge is the codename for micro-architecture based on the 32 nm manufacturing process developed by Intel to replace Westmere micro-architecture. Due to Advanced Vector Extensions (AVX) 256 bits instruction set with wider vectors, a full socket of phinally system has overall theoretical peak performance P_{peak} of 172.8 GFlops/s and 345.6 GFlops/s for double and single

precision, respectively at base clock speed:

$$\begin{aligned} P_{peak} &= n * F * S * f \\ &= 8 * 2 * 4(DP) \ or \ 8(SP) * 2.7 \\ &= 172.8 GFlops/s(DP) \qquad or \qquad 345.6 GFlops/s(SP) \end{aligned} \tag{3.1}$$

where n is the number of cores, F is the floating point instruction per cycle (i.e, one addition and one multiplication), S is the floating point operations per instruction and f is the clock speed.

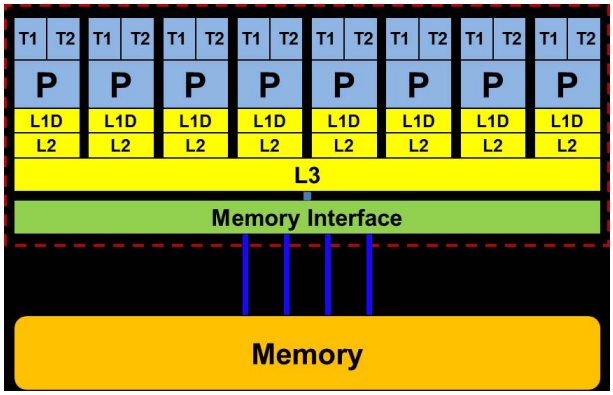


Figure 3.2.: Intel eight cores Sandy Bridge EP processor Socket

Important features of the Intel Sandy Bridge micro-architecture are [6], [7]:

- Sandy Bridge has Intel's second-generation core processor technologies.
- Sandy Bridge core can sustain one full-width AVX load and one half-width AVX store per cycle for 48 bytes/cycle as load throughput of the L1 cache is doubled from 16 to 32 bytes per cycle with double precision AVX instruction against SSE instruction of its predecessor, Westmere.
- The core can execute one addition and one multiplication instruction per cycle.
- The L3 cache is segmented, with one segment per core. All segments of L3 cache have the same bandwidth capabilities as the L2 cache (i.e., 256 bits per cycle), so streaming loop kernels provide a good scaling behaviour when the data comes from L3 cache.
- Sandy Bridge has a 256 bit/cycle ring bus interconnect between cores, graphics, cache and system agent domain.
- A full 64-byte cache line read or write takes two cycles.
- All components of Sandy Bridge run at the same clock frequency, which can be set to a fixed value in the range from 1.2 - 2.7 GHz.
- Thermal design power, TDP, of Sandy bridge processor at 2.7 GHz base clock speed is 130 Watts.
- Sandy Bridge has 64 GB (DDR3-1333) memory module per node for a peak bandwidth of 42 GB/s.
- The clock speed of the DRAM module is constant and independent of the core clock.

3.2. “ivyep1” Testsystem

The “ivyep1” test system contains a dual-socket Intel 10 cores Ivy Bridge EP processor with 20 logical cores per socket through simultaneous multi-threading. It operates at 3.0 GHz base clock speed and features Intel turbo boost technology by allowing processors to operate above the rated frequency. The codename for the successor to Sandy Bridge micro-architecture is Ivy Bridge which is based on the 22 nm manufacturing process developed by Intel. A full socket of “ivyep1” system has the overall theoretical peak performance P_{peak} of 240 GFlops/s and 480 GFlops/s for double precision and single precision, respectively:

$$\begin{aligned} P_{peak} &= n * F * S * f \\ &= 10 * 2 * 4(DP) \ or \ 8(SP) * 3 \\ &= 240 GFlops/s(DP) \qquad or \qquad 480 GFlops/s(SP) \end{aligned} \tag{3.2}$$

Ivy Bridge-EP chips include following significant changes over Sandy Bridge [6], [7]:

- Ivy Bridge-EP has Intel’s third-generation core processor technologies instead of second-generation.
- “Tick-Tock” is a model adopted by Intel chip manufacturer in which every “tick” is a shrinking of process technology of the previous micro-architecture and every “tock” is a new micro-architecture. Ivy Bridge is a “tick” and Sandy Bridge was “tock” as die shrink from 32 nm to 22 nm (see Fig. 3.1).
- Intel has added some additional features in Ivy Bridge as well, such as, more advanced integrated graphics and a new type of three dimensional “Tri-gate” transistor in order to reduce die size.
- Performance generally improves more between “ticks” and “tocks” than between “tocks” and “ticks”. Thus, compared to its predecessor, Ivy Bridge is expected to offer a small improved performance over Sandy Bridge.
- Ivy Bridge-EP and its predecessor Sandy Bridge are backward-compatible i.e. Sandy Bridge processors work in Ivy Bridge motherboards, and vice versa with upgraded motherboard’s BIOS.
- All components of Sandy Bridge processor run at the same clock frequency, which can be set to a fixed value in the range from 1.2 - 3.0 GHz.
- With die shrinks, Ivy Bridge processors generally come with reduction in power consumption than Sandy Bridge processors under load. Detail analysis of its power consumption will come in Section 6.4.
- Ivy Bridge-EP has 64 GB (DDR3-1866) memory module per node for a theoretical peak bandwidth of about 48 GB/s.

3.3. “hasep1” Testsystem

The “hasep1” test system has a dual-socket Intel 14 cores Haswell EX processor with 56 logical cores through hyper-threading per node. It operates at 2.3 GHz base clock speed and also features Intel Turbo mode for increased performance. Haswell is the codename for a processor micro architecture developed by Intel as the successor to the Sandy Bridge and Ivy Bridge architecture. Its architecture is specifically designed to optimize the power savings and performance

benefits from the move to "FinFET non-planar 3D transistors" on the improved 22 nm process node. Haswell microprocessors represents the "tock" in Intel's CPU development program. The "hasep1" system has overall theoretical peak performance P_{peak} of 515.2 GFlops/s and 1030.4 GFlops/s per socket for double precision and single precision, respectively at base clock speed.

$$\begin{aligned} P_{peak} &= n * F * S * f \\ &= 14 * 4 * 4(DP) \ or \ 8(SP) * 2.3 \\ &= 515.2 GFlops/s(DP) \qquad or \qquad 1030.4 GFlops/s(SP) \end{aligned} \tag{3.3}$$

Important features of Haswell microprocessor architecture over Sandy Bridge and Ivy Bridge are as [18]:

- Haswell is Intel's fourth-generation core microprocessor family instead of second or third-generation.
- The execution units in Haswell are vastly improved over previous generations, particularly to support AVX2 and the new fused multiply add (FMA). Generally, the SIMD integer performance doubled due to wider 256-bit AVX2 instructions. The theoretical peak floating point performance has basically doubled by virtue of the 256-bit FMA instructions i.e. extra add in the FMA is essentially free from a latency perspective, yielding 16 DP FLOP/cycle for each Haswell core.
- Haswell adds advance features as well e.g., an integer dispatch port so it can handle many instructions while the SIMD dispatch ports are fully utilized, and a new memory port.
- The memory hierarchy for Haswell is probably the biggest departure from the previous generation. The whole memory system has been enhanced to support gather instructions and transactional memory.
- More significantly, the cache hierarchy has same capacity, organization, and latency but can sustain twice the bandwidth, two 256-bit loads and a 256-bit store per cycle, for 96 Bytes/cycle compared with 48 Bytes/cycle for Sandy Bridge.
- A full 64 Bytes cache line can be read each cycle.
- All components of Haswell processor run at the same clock frequency, which can be set to a fixed value in the range from 1.2 - 2.3 GHz.
- 64 GB main memory per Haswell EP processor node for a theoretical peak bandwidth of about 60 GB/s.

Memory Hierarchies of Sandy Bridge, Ivy Bridge and Haswell architectures

Table 3.1 summarizes the cache topology of different generation of Intel's micro-processors [9], [19]. Moreover, All Intel's design caches uses 64 byte cache lines and single ported L1 cache (that is, L1 cache cannot communicate with the registers and L2 cache at the same time).

Table 3.1.: Memory Hierarchies of Sandy Bridge, Ivy Bridge and Haswell

Metrics	Sandy Bridge	Ivy Bridge	Haswell
Core frequency	2.7 GHz	3 GHz	2.3 GHz
Cores per node	8	10	14
Peak DP FP performance	172.8 GFlops/s	240 GFlops/s	515.2 GFlops/s
Peak SP FP performance	345.6 GFlops/s	480 GFlops/s	1030.4 GFlops/s
L1 instruction cache	32 KB	32 KB	32 KB
L1 data cache	32 KB	32 KB	32 KB
Load Bandwidth	32 Bytes/cycle	32 Bytes/cycle	64 Bytes/cycle
Store Bandwidth	16 Bytes/cycle	16 Bytes/cycle	32 Bytes/cycle
Total Bandwidth	48 Bytes/cycle	48 Bytes/cycle	96 Bytes/cycle
L2 unified cache	256 KB	256 KB	256 KB
Bandwidth to L1	32 Bytes/cycle	32 Bytes/cycle	64 Bytes/cycle
L3 shared cache	20 MB	25 MB	35 MB
Memory	64 GB (DDR3-1333)	64 GB (DDR3-1866)	64 GB
Memory Bandwidth	∼42 GB/s	∼48 GB/s	∼60 GB/s

3.4. “emmy” Compute-Cluster

RRZE production cluster “emmy” was installed in 2013 and was named after famous German mathematician Amalie Emmy Noether who was born in Erlangen. It is designed for running massively parallel programs using significantly more than one node and high speed interconnect. It has overall theoretical peak performance P_{peak} of 197.12 TFlop/s and LINPACK Performance P_{max} of 191.5 TFlop/s [4].

$$\begin{aligned} P_{peak} &= N_{nodes} * n * F * S * f \\ &= 560 * 20 * 2 * 4(DP) \ or \ 8(SP) * 2.2 \\ &= 197.12 TFlops/s(DP) \qquad or \qquad 394.24 TFlops/s(SP) \end{aligned} \tag{3.4}$$

where N_{nodes} and n are the number of nodes and cores, respectively.

Moreover, emmy has rank 407 in Top500 list published in June 2015, which rank parallel computers based on their performance in the LINPACK benchmark [4]. Despite of the increased degree of parallelism and peak performance of the complete emmy systems compare to previous systems, non-vectorised serial code can neither benefit from the wider SIMD units nor from the increased number of cores per node but suffers from the decreased clock frequency.
Short overview of emmy compute cluster is as following.

- Vendor: NEC (Dual-Twin Supermicro)
- Operating system: CentOS (Redhat Enterprise without support)
- SIMD vector length: 256 bit (AVX)
- Power consumption: 169 KW (back-door heat exchanger)
- Compiler: Intel, GCC and others in various versions
- Math Library: Intel MKL
- 2 frontend nodes with the same CPUs (“emmy1” and “emmy2”)

Hardware Topology

The RRZE "emmy" cluster consists of total 560 compute nodes with 11200 physical cores and 22400 logical cores through hyper-threading. Out of total 560 compute nodes, there are 544 regular compute nodes and 16 accelerator nodes. The command "pbsnodes -a | grep '^e' | wc -l" searches and counts all available nodes. Their further hardware topology is as follows:

- 544 regular compute nodes
  - each node has dual-socket ten cores Intel Xeon E5-2660v2 "Ivy Bridge EP"processor with hyper-threading (i.e. 20 physical cores per node showing up as 40 virtual cores per node). It operates at 2.2 GHz base clock speed and also include Intel turbo mode feature.
  - each with 64 GB DDR3-1600 memory modules per emmy node
  - each with NO local disks
  - All 544 nodes distributed on 11 Racks [9], [19]
    - Rack 1 with 56 compute nodes
    - Rack 2 with 52 compute nodes
    - Rack 3 with 56 compute nodes
    - Rack 4 with 52 compute nodes
    - Rack 5 with 52 compute nodes
    - Rack 6 with 56 compute nodes
    - Rack 7 with 52 compute nodes
    - Rack 8 with 52 compute nodes
    - Rack 9 with 52 compute nodes
    - Rack 10 with 32 compute nodes
    - Rack 11 with 32 compute nodes
- 16 nodes with accelerators
  - 8 nodes with 2 x NVIDIA K20 GPGPUs
  - 8 nodes with 2 x Intel Xeon Phi coprocessors
  - 64 GB DDR3-1866 memory modules per node
  - 1 TB local hard disk
  - All 16 accelerator nodes distributed on 2 Racks [9], [19]
    - 8 nodes on Rack 10
    - 8 nodes on Rack 11

Interconnect Network

The network on emmy is a full Quad Data Rate (QDR) Infiniband interconnect fabric. It is fully non-blocking and each link has 40 Gbit/s bandwidth per direction.

3.5. Measurement methodology

3.5.1. Module and compiler

On RRZE HPC systems, a modules environment is provided to facilitate access to software packages and the "module avail" command delivers a list of all available packages. However, Intel compilers (i.e., ifort (Fortran77/90), icc (C) and icpc (C++)) were used for present work, since they provide higher performance than the GCC. To set up a PATH with the Intel compilers, a "module load intel64" command was executed once per shell, where the Intel compiler with specify a version number "Intel64" sets the necessary setting for 64 bit systems. Intel compiler command line option -O3 -xAVX -fno-alias was preferred for current work. Here is the makefile for 2D-Jacobi stencil benchmark code:

```
LIKWID=${LIKWID_INC} -DLIKWID_PERFMON ${LIKWID_LIB} -llikwid -pthread
RZOMP=-openmp

jacobi2d.exe: relax_line.h relax_line.c jacobi2d.c
icc ${RZOMP} -O3 -xAVX -fno-alias -vec-report3 -restrict -fno-inline relax_line.c jacobi2d.c -o jacobi2d.exe ${LIKWID}

asm: relax_line.c
icc -O3 -xAVX -fno-alias -vec-report3 -restrict -fno-inline -fsource-asm -S relax_line.c -o relax_line.s

clean:
rm -rf jacobi2d.exe *.o
```

The makefile for DGEMM benchmark is shown in Appendix C.1.

3.5.2. Batch Scripts and LIKWID Tools

All resources of HPC production clusters are controlled through a batch system. The user jobs with a shell script are submitted through the batch system to the cluster as:

```
qsub -l nodes=<nnn>:ppn=xx,walltime=HH:MM:SS script.sh
```

These are further details and explanation of this queue:

qsub: The command for job submission, qsub , provides the job ID, standard output file and error output file which can later be used for identification purposes. File name for the standard output or error stream is compiled from batch script file name and the job ID.

Further options: As always, full nodes have to be requested on qsub:

```
-l nodes=<nnn>:ppn=xx,walltime=HH:MM:SS
```

These Job parameters specifies the required number of nodes/CPUs "<nnn>", the number of hardware threads per nodes "xx" and the estimated wall clock runtime "HH:MM:SS" respectively. For example, the number of threads per nodes need to be specified as ppn=40 for "emmy", ppn=32 for "phinally" and ppn=56 for "hasep1".
Whenever a particular node has to be required, it is requested as

```
-l nodes=exxxx:ppn=xx,walltime=HH:MM:SS
```

Whenever a specific node type is required, it is requested by affixing ":property" with above mentioned queue. For example, on "emmy" cluster, different memory modules and likwid properties are requested as:

```
-l nodes=<nnn>:ppn=40:ddr1600,walltime=HH:MM:SS to qualify 544 nodes
-l nodes=<nnn>:ppn=40:ddr1866,walltime=HH:MM:SS to qualify 516 nodes
-l nodes=<nnn>:ppn=40:likwid,walltime=HH:MM:SS to use likwid
```

Script file: A bash-based shell script file contains all executable commands.

```
1  #!/bin/bash -l
2  cd $PBS_O_WORKDIR
3  module load likwid intel64
4
5  FREQ=`likwid-setFrequencies -l | sed -e 's/Available frequencies: //'`
6  for f in $FREQ; do
7  likwid-setFrequencies -f $f
8
9  # iterate thread count on each socket in turn
10 for s in S0 S1; do
11 for t in `seq 0 9`; do
12 likwid-perfctr -m -C ${s}:0-${t} -g ENERGY ./jacobi2d.exe 8000 8000 0 0 100 > ↩
       RESULT_${HOSTNAME}_${f}_${s}_${t}.dat
13 done
14 done
15 done
```

Here in line 1, "-l" switch is used to load modules from the inside of a batch script. All batch scripts are start executing in the user's $HOME directory so line 2 is used to change it to the directory where the job was submitted.

likwid-setFrequencies tool: For getting accurate timings in terms of processor cycles, i need to fix the clock speed of the CPU, which is done by using likwid-setFrequencies tool [9], [19]. The line 5 delivers a list of supported frequencies for a system and a specific frequency is set in line 7. In this script, the clock speed of all cores on each socket is sequentially set to all available CPU clock frequencies. For example, for emmy with 2.2 GHz base clock speed, it selected from the following options: f2.201, f2.2, f2.1, f2.0, f1.9, f1.8, f1.7, f1.6, f1.5, f1.4, f1.3, f1.2. However, the turbo mode "f2.201" feature was deliberately ignored for present work.

likwid-pin tool: Correct pinning of the OpenMP threads is essential to achieve optimum performance and also to avoid the devastating effects of ccNUMA machines. Pinning is done by using likwid-pin tool [9], [19] which enforced thread-core affinity in a multi-threaded application without touching the source code and also set the environment variable "OMP_NUM_THREADS" automatically. As default, physical numbering of the cores given by likwid-topology tool [9], [19] is set, but also logical numbering inside the node (e.g., via N prefix) or inside the sockets (via S# prefix, e.g., S0 or S1) can be used. Physical cores come first in logical numbering. In the present script, the cores are logical numbered over the whole node and physical cores come

first with using a N prefix (-c N:0-9). For example, emmy has 10 cores with 20 SMT threads per socket, so all physical cores can obtain with "-c N:0-9" and all physical cores and SMT threads can access via "-c N:0-19".

likwid-perfctr tool: likwid-perfctr [9], [19] measures performance counter metrics over the complete application runtime or between arbitrary points in the code with API support. It includes all functionality of likwid-pin for pinning a threaded application and possible to specify the full hardware-dependent event names with existing pre-configured event sets. These groups provide useful event sets and compute common derived metrics. However, the uncore counters measure per socket. Therefore, likwid-perfctr has a socket lock which ensures that only one thread per socket starts the counters and only one thread per socket stops them. The first thread arriving in start or stop gets the lock. The following illustrates the use of the marker API in a Jacobi stencil code with named region "J2D".

```
1 #include <likwid.h>
2 LIKWID_MARKER_INIT;
3 #pragma omp parallel{
4 LIKWID_MARKER_THREADINIT;
5 }
6 #pragma omp parallel{
7 LIKWID_MARKER_START("J2D");
8 }
9 !.... measured code region "J2D" here ....
10 #pragma omp parallel{
11 LIKWID_MARKER_STOP("J2D");
12 }
13 LIKWID_MARKER_CLOSE;
```

likwid header is included in line 1 to ensure compilation of 2D-jacobi stencil benchmark code even with unavailability of LIKWID tool. It contains a set of macros which allowed activation of the marker API by defining LIKWID_PERFMON. The function calls in line 2 and 13 (i.e., LIKWID_MARKER_INIT and LIKWID_MARKER_CLOSE) are done from the serial part of the application but routines, such as, LIKWID_MARKER_THREADINIT (line 4), LIKWID_MARKER_START("J2D") (line 7) and LIKWID_MARKER_STOP("J2D") (line 11) are called in a parallel region for each thread.

```
1 likwid-perfctr -m -C ${s}:0-${t} -g ENERGY ./jacobi2d.exe
```

In above mentioned script command, likwid-perfctr for a threaded OpenMP jacobi application is compiled with pinning and measures the performance group "ENERGY" between arbitrary points in jacobi benchmark with API support. The "OMP_NUM_THREADS" command is omitted so it is set by likwid automatically. Flag -C configures the core ids of the measured counters and flag -g specified measures group or event (e.g., ENERGY in this case). The output consists of different tables with the raw event counts, with derived metrics and with statistical data as minimum, maximum and mean. For gathering processor specific information about hardware performance capabilities and groups, "-a", "-g" and "-H" switches are used. For example, "likwid-perfctr -a" prints available groups on an architecture.

```
1 likwid-perfctr -m -C ${s}:0-${t} -g ENERGY ./jacobi2d.exe 8000 8000 0 0 100 > ↩
    RESULT_${HOSTNAME}_${f}_${s}_${t}.dat
```

Shortly, the power and energy characteristics of an OpenMP-parallel 2-D Jacobi stencil solver is analysed on all available threads of both sockets in turn with working set “isize”, “jsize”, “iblocksize” and “jblocksize” of 8000 and 100 sweeps. It measures the performance group “ENERGY” and delivers “CPU”, “DRAM” and “PP0” power and energy measurements. The results are to stdout with specifying “.dat” as suffix. Following script is used for analysis of these outcomes individually with a variation in subsequent number of cores and frequency per socket.

```
#!/bin/bash -l

# List available frequencies
FREQ=`likwid-setFrequencies -l | sed -e 's/Available frequencies: //'`

# iterate thread count on each socket in turn
for s in S0 S1; do
for n in seq 0 9; do
for f in $FREQ; do

# Set frequencies
likwid-setFrequencies -f $f

grep "Power DRAM" ./RESULT_exxxx_${f}_${s}_${n}.dat >> Power_DRAM.dat

done
done
done
```

To graph a specific parameter just “grep” that parameter from text file. For example, here grep “Power DRAM” searches the input file named RESULT_exxxx_${f}_${s}_${n} for lines containing a match to the Power DRAM and prints the matching lines to another file name Power_DRAM.dat. Also, search can either be done for other parameters of “ENERGY” performance group (e.g., CPU power, CPU and DRAM energy etc.) or of “MEM” performance group (e.g., memory bandwidth etc).

The event monitoring counters of modern processors count various processor-internal events correspond to activities on the processor chip and allow estimating the power/energy consumption of the memory controller and CPU-level components listed in Table 3.2.

Table 3.2.: List of available Power/Energy counters

Counters	Description
Power(/Energy)_PKG	Power/Energy consumption of whole CPU package
Power(/Energy)_PP0	Power/Energy consumption of Processor cores, L1 and L2 caches
Power(/Energy)_DRAM	Power/Energy consumption of memory controller

3.5.3. Levenberg-Marquardt Non-linear fitting algorithm

The Levenberg-Marquardt (LM) curve-fitting method [20] is a standard technique used to solve non-linear least-squares problems, and is employed to get w_i parameters of analytical CPU power model. It involves an iterative improvement of w_i parameter values that locates the minimum

of the CPU power function expressed as the sum of squares of non-linear functions. Non-linear least squares problem arises when fitting a parametrized non-linear power function to a set of measured data points by minimizing the sum of the squares of the errors between the data points and the non-linear power function.

Levenberg-Marquardt vs. Steepest descent and Gauss-Newton algorithm

The Levenberg-Marquardt curve-fitting method [20], [21] uses a search direction that is a combination of both the steepest descent and the Gauss-Newton (GN) minimization method. In the steepest descent method, the sum of the squared errors is reduced by updating the parameters in the direction of the greatest reduction of the least squares objective. Whereas, in the Gauss-Newton method, the sum of the squared errors is reduced by assuming the least squares function is locally quadratic.

When the w_i parameters are far from their optimal value, the Levenberg-Marquardt algorithm behaves like a steepest descent method i.e. slow, but guaranteed to converge; however, when close to their optimal value, it acts like a Gauss-Newton method [21]. For well-behaved functions and reasonable starting w_i parameters, the LMA tended to be a bit slower than the Gauss-Newton that means Gauss-Newton method is generally more effective when the residual is zero at the solution. However, the Levenberg-Marquardt method is more robust than the Gauss-Newton method i.e. in many cases it finds a solution even if it start very far off the final minimum.

Levenberg-Marquardt algorithm

Levenberg-Marquardt is a popular method of finding the minimum of a function f(x) that is a sum of squares of nonlinear functions,

$$\min_x f(x) = \|F(x)\|_2^2 = \min_x \sum_i F_i^2(x), \tag{3.5}$$

where the residual $\|F(x)\|$ is smallest at the optimum since realistically achievable target trajectories are established. Let the Jacobian matrix of F(x) be denoted J(x), then the Levenberg-Marquardt method searches in the direction given by the solution of the linear set of equations

$$(J(x_k)^T J(x_k)) + \lambda_k I)d_k = -J(x_k)^T F(x_k), \tag{3.6}$$

where I is the identity matrix, $J(x_k)^T J(x_k)$ is approximate hessian and the non-negative damping factor λ_k controls both the magnitude and direction of d_k.

With zero value of damping factor λ_k, the direction d_k is identical to that of the Gauss-Newton method. However, when daming factor λ_k tends to infinity, the direction d_k tends towards the steepest descent direction with magnitude tending to zero implying that the term $F(x_k + d_k) < F(x_k)$ holds true. Thus, for the next value of damping factor λ_{k+1}, the algorithm divide current damping factor value by ten. However, when the step is unsuccessful, the algorithm sets λ_{k+1} by multiplying its current value with ten [21], [22].

Levenberg-Marquardt MATLAB code

A Matlab code was developed for plotting data-fitting of nonlinear Power function with frequency variation using Levenberg-Marquardt algorithm. Matlab has built-in Levenberg-Marquardt algorithm which provides a fitresult (i.e, a fit object representing the fit) variable and gof (i.e, a structure with goodness-of fit info) variable (see Appendix B.1).

Chapter 4

Performance, Power and Energy Characteristics of Benchmark Codes

Recently, the increased microprocessor complexity and frequency have caused the power consumption to grow to the level where power has become a first-order issue. Therefore, in this chapter, the properties of different benchmarks will be investigated from the performance, the power dissipation and the energy consumption perspective.

To obtain predictive models for system power/energy, the actual measurements were analysed from characterization experiments on selective benchmark codes. All the measurements and resulting curves of benchmark codes were performed on the phinally machine (i.e, 8 cores Intel Sandy Bridge processor with 2.7 GHz base clock speed), which operated at varying number of active cores and clock frequency. All measurements were obtained after a "warm-up" phase to have steady state values. The "warm-up" phase means that we have run the code N seconds before starting the actual measurement. The following first two benchmarks (see Section 4.1,4.2) were selected specifically as they represent a typical "corner case" scenario (i.e., perfectly scalable and saturating cases) for loop-based scientific computing and third benchmark (see Section 4.3) was used as it is a complete algorithm that actually does something useful.

4.1. Dense matrix-matrix multiplication (Core-bound case)

A dense matrix-matrix multiply, DGEMM, is a corner case for applications that are almost perfectly scalable. Here, the function of DGEMM code is to implement the multiplication of two dense double precision matrices of size 8000^2, which results in 1.02 GB data and fits into memory. This code is a "pure compute" case (i.e., scalable with speed up of 7.78 on eight cores) and implemented using thread parallel Intel MKL math library (see Appendix C.1).

The performance, power and/or energy related data of this DGEMM code tabulated in Table 4.1 shows that it runs very close to peak performance of phinally machine (i.e., 91% of arithmetic peak performance) which makes it the hottest code among all these studied benchmarks with highest on-chip power and energy consumption. Furthermore, for applications which are core-bound, we can say that although the data comes from memory but due to blocking and unrolling in core, pressure on memory interface is not high so memory bandwidth is not saturating which causes lower DRAM power consumption compare to memory-bound applications.

Table 4.1.: The performance, power and/or energy related data of benchmark codes on 8 cores of Sandy Bridge processor at 2.7 GHz base clock speed

Metrices	DGEMM	Jacobi
Performance (P_{max})	157.3 GFlops/s	6.2 GFlops/s
Cycles per Instruction (CPI)	0.35	7.8
Instruction per cycles (IPC)	2.86	0.128
Memory Bandwidth	6 GB/s	38 GB/s
CPU power (W^{CPU})	108.4 W	91.3 W
DRAM power (W^{DRAM})	14.6 W	35.5 W
Total power (W^{Total})	123 W	126.8 W
CPU energy (E^{CPU})	7218 J @ 157.3 GF/s	375 J @ 6.2 GF/s
DRAM energy (E^{DRAM})	1422 J @ 157.3 GF/s	146 J @ 6.2 GF/s
Total energy (E^{Total})	8640 J @ 157.3 GF/s	521 J @ 6.2 GF/s

4.2. 2D Jacobi stencil (Memory-bound case)

An OpenMP-parallel 2D Jacobi stencil solver (see Appendix C.2) is defined as a function that updates a point based on values of its neighbours and is a corner case for applications that gets limited by off-chip bandwidth. An AVX-vectorized Jacobi code performs 100 sweeps through 8000^2 double precision lattice sites data structure that is much larger (1.02GB data set) than the capacity of the available data caches in Sandy Bridge processor. This chosen configurable problem size has a well-defined layer condition [23] for L3 cache (i.e., $3*n*imax*8B < \frac{L3Cache}{2}$).

Overall performance of jacobi stencil computation is memory-bound, since its code balance B_c is much larger than the machine balance B_m. Beyond saturation point, roofline model is used to analyse and predict the performance of Jacobi stencil [11]. For each lattice site update (LUP), a Jacobi stencil will perform 4 floating point operations for every 24 bytes of memory traffic on write-allocate architecture, implying a code balance of 6 B/F. Roofline model can predict performance by ratio of saturated memory bandwidth ,b_s, to effective code balance B_c in non-saturated region (i.e., $P_{roof} = min(P_{max}, \frac{b_s}{B_c})$). The computational intensity I_c of 0.167 F/B is the inverse of code balance B_c and reflects the number of floating point operations transferred from memory . Hence, expected saturation performance calculated by roofline model [11] will be 1666 MLUP/s or 6.6 GF/s on a full Sandy Bridge chip, which matches the actual measurement result quite well (see Table 4.1).

These measurements reflect that jacobi solver runs at 3.6% of arithmetic peak performance of phinally machine so the memory-bound codes runs vary far from peak performance of operating system. Moreover, due to memory bandwidth bottleneck, they demonstrate a relatively higher dynamic random access memory (DRAM) power and energy consumption compare to compute bound applications.

4.3. Conjugate Gradient Method

An OpenMP parallel Conjugate Gradient Method is used as an iterative method of choice to solve a large sparse systems of linear equations due to its low memory requirements and exact convergence in a finite number of iterations. The CG algorithm is better than other iterative

solvers like Jacobi Method, since the solution is obtained in fewer iterations. It has been analysed for two different sizes of real symmetric positive definite matrices for comparison; First is steady state thermal problem, named as Schmid/thermal2, with out-of-cache data storage of 8.58 M * (8 B + 4 B) = 0.1 GB (matrix dimensions of $1,228,045^2$ and 8,580,313 number of non-zeros entries, N_{nnz}). Second is 27-point stencil matrix, Hpccg-100, with 1 M^2 matrix size and 26,463,592 N_{nnz} entries. It results in 0.3 GB data set. Schmid/thermal2 problem converges in 5720 iterations, whereas Hpccg-100 provides solution in 56 iterations for prescribed tolerance of 10^{-10} without any preconditioning.

The CG algorithm is compose of couple of kernels, such as, it has three vector updates, two inner products, one sparse matrix vector product (spMVM) and two sequential parts (see Appendix C.3). By analysing the code the most time consuming step of algorithm was found to be sparse matrix-vector multiplication, which has a complexity of order of n. The performance of the CG algorithm is mainly dominated by the efficiency of implementation of this sparse matrix-vector product.

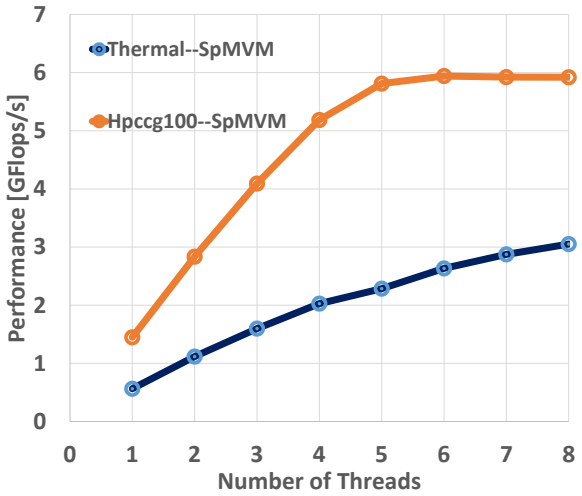


Figure 4.1.: Performance of spMVM kernal in CG Method for "Schmid/Thermal2" and "hpccg-100" matrices at 2.7 GHz base clock speed of "phinally" system

While looking at the performance it was observed that spMVM for Thermal2 matrix does not seem to be saturating compare to Hpccg-100 which is a well defined matrix (see Fig. 4.1) with a performance different of about a factor of 2 (see Table 4.2).The reason for this non-saturating behaviour of a matrix might be that the matrix may fit into cache which does not apply in the present case as both matrices are really large and fit into memory. This non-saturating behaviour of spMVM in the conjugate gradient method can be explained by following three reasons:

1. There exists some load-imbalance, since number of non-zeros per row, N_{nzr} are not constant across the matrix rows. This problem can be fixed by using static OpenMP schedule with a suitable chunk size.
2. The memory bandwidth drawn by the spMVM with the "Thermal2" matrix was significantly lower than the maximum (i.e., 27 GB/s memory bandwidth against saturated bandwidth of the Sandy Bridge architecture), which is justified by some latency effect. One can't do much about latency effect because of scattered structure of matrix. Further, if a matrix is really scattered, so every time it gets a complete cache line with accessing

only one element for right hand side vector and results in wastage of the bandwidth. In addition, when one jumps from cache line to cache line, pre-fetchers does not work. The only way to fix it is to reorder matrix with non-zero's closer to diagonal, as the case with Hpccg-100 matrix. The main reason why Hpccg-100 matrix saturates the bandwidth is that it has a completely regular structure (dense subdiagonals), i.e., the memory access is streaming type. A well defined matrix drives lower traffic for loading right hand side vector (i.e., alpha goes down) and the pre-fetchers work better with more organize access.

3. Alpha effect for spMVM [24], which quantifies traffic for loading right hand side vector, is dependent on number of threads. The more threads used, the less cache and the less reuse potential will obtain per threads with simple static decomposition. There seems to appear some alpha effect with "Thermal2" matrix as loading more data with minimum actual data.

In conjugate gradient method, first step is sparse matrix-vector multiplication and all other steps come afterwards. As first step for spMVM already streams a big matrix to cache; thus there is no space left for other steps. Hence, other scalar product steps are saturated and similarly all left vector operations with 3 vectors each of 1.2M or 1M entries leads to memory bound characteristics of vector operations. In the end, the CG algorithm for "Thermal2" matrix is a mixture of scalable and saturated kernels because of non-saturating behaviour of spMVM kernel; whereas "Hpccg-100" matrix is purely compose of memory-bound kernels.

Table 4.2.: The performance, power and/or energy related data of spMVM step in CG method for both "Thermal2" and "Hpccg-100" problems on 8 cores of Sandy Bridge processor at 2.7 GHz base clock speed

Metrices	Thermal2-spMVM	Hpccg100-spMVM
Performance (P_{max})	3.05 GFlops/s	5.92 GFlops/s
Memory Bandwidth	27 GB/s	38 GB/s
Iteration count for convergence	5720	56
CPU power (W^{CPU})	83.15 Watts	88.34 Watts
DRAM power (W^{DRAM})	30.35 Watts	31.26 Watts
Total power (W^{Total})	113.5 Watts	119.6 Watts
CPU energy (E^{CPU})	2674.5 Joules	44.05 Joules
DRAM energy (E^{DRAM})	976.29 Joules	15.67 Joules
Total energy (E^{Total})	3650.8 Joules	59.72 Joules
Total energy per flop per iteration (E^{TOTAL}_{Flop})	37 nJoules	20 nJoules

The overall energy consumption of an application is affected by the total amount of work done. Each iteration of "Hpccg-100" matrix (i.e., 26M number of non-zeros N_{nnz}) does about a factor of 3 more work per sparse matrix-matrix multiplication against "Thermal2" matrix (i.e., 8.5M number of non-zeros N_{nnz}). On the other hand, the "Thermal2" matrix has a much higher number of iteration required for convergent solution (i.e., 5720 versus 56 number of iterations). Hence, an overall energy consumption per flop value for one iteration is calculated for comparison of both studied matrices.

Hence, the studied benchmark applications can organized according to their computational intensities. The maximum intensity for spMvM kernel is 1/6 flop/bytes and the Hpccg-100

matrix is at this limit.

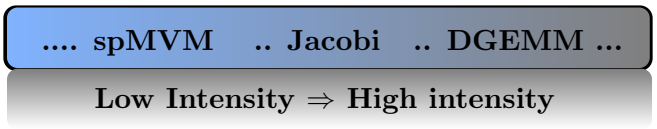


Figure 4.2.: The order of studied benchmarks with respect to their respective computational intensities

4.4. Frequency and cores variation effect on power/energy characteristics

In a system, two variables have a significant effect on the power draw of an individual active machine. First is the CPU frequency, which also affects core voltage and second is the number of cores utilized. Other influence factors such as code characteristics and chip temperature have a measurable but smaller impact. Therefore, an exploration of the interesting properties of both multi-core CPU and DRAM power dissipation and energy consumption are described with respect to clock frequency and number of utilized cores by studying selective benchmark codes.

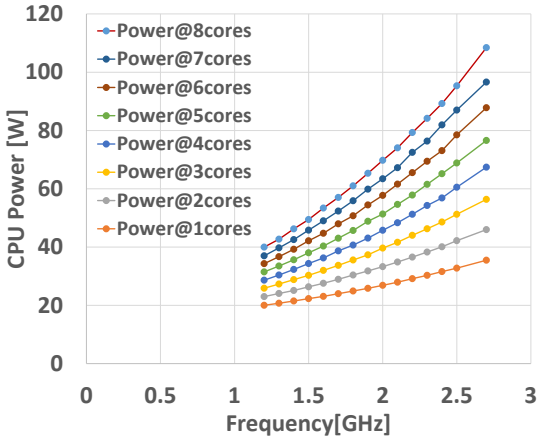
Variation effect on Power dissipation

Actual measurements of the dynamic on-chip power dissipation are in line with results of Hager et al. [10] that it is a quadratic polynomial in the clock frequency, f, and has a linear dependence on the number of active cores, n, in non-saurated region for any fixed f and n, respectively. Moreover, when applications get limited by off-chip bandwidth, the slope of the power dissipation decreases slightly without providing any performance improvement because the cores start become idle while waiting for memory (see Fig. 4.3).

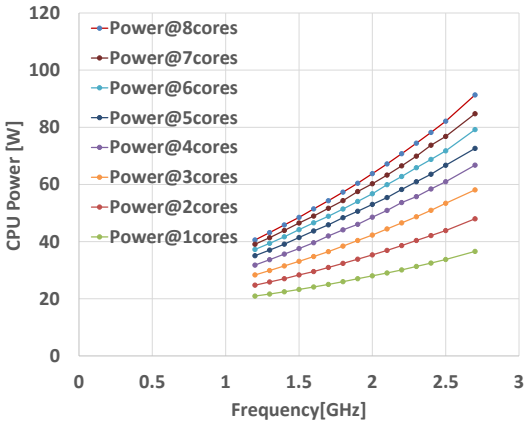
Measurements of the system DRAM power dissipation depending on workloads (memory-bound vs. cache-bound) with variant cores and frequency configurations illustrates that it correlates with performance and is linear in clock speed and number of threads until it hits a bottleneck. However, beyond saturation, DRAM power remains maximum (see Fig. 4.4).

In Fig. 4.4(c,d), for 1 core of compute-bound DGEMM case, expected perfect linearity of the DRAM power dissipation in clock frequency is not visible, since very low pressure on memory interface implies a negligible amount of memory bandwidth for one core (0.83 GB/s @1core) such that it does not play any significant role. Whereas, for memory-bound jacobi case, even for 1 core, perfectly linearity in frequency can observe because of substantial amount of memory bandwidth (12 GB/s @1core).

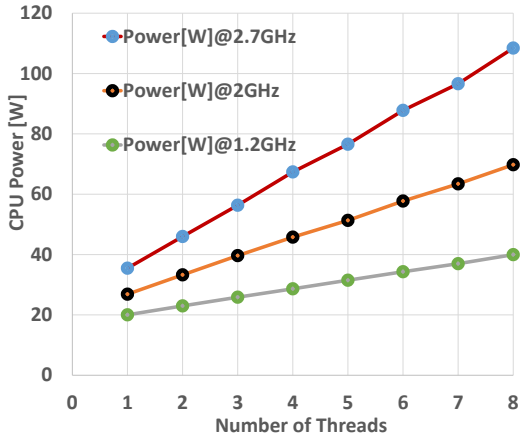
For memory bandwidth limited Jacobi benchmark case, instead of constant DRAM power, a drop at lower frequency is observed when it gets limited by bottleneck (see Fig. 4.4(d)). This falling curve might be caused due to two reasons: Bandwidth dependency on clock frequency or non-saturating performance at lower frequency (see Fig. 4.4(b)). For example, at 5 cores, falling curve of DRAM power is justified by second reason that it saturates at base frequency but no more saturating at lower frequency. Similarly, for 8 number of cores, a drop of DRAM power dissipation is justified by bandwidth reduction at lower frequency. The DRAM bandwidth is not linear in the core clock speed even of the code saturates at all frequencies. The exact reason for this peculiar effect is not clear, but we know that it is gone on Haswell (i.e., on Haswell, the memory BW is almost insensitive to core clock speed).



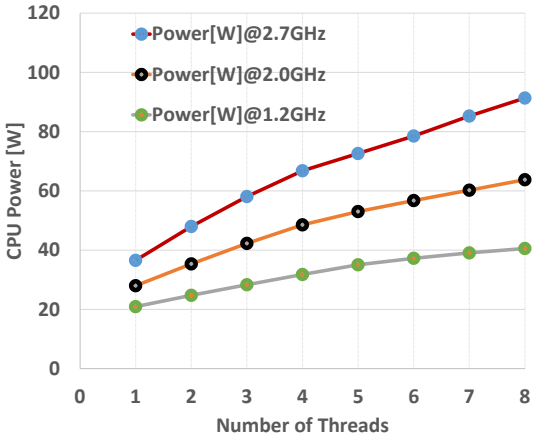
(a) DGEMM Benchmark Code



(b) Jacobi Benchmark Code



(c) DGEMM Benchmark Code



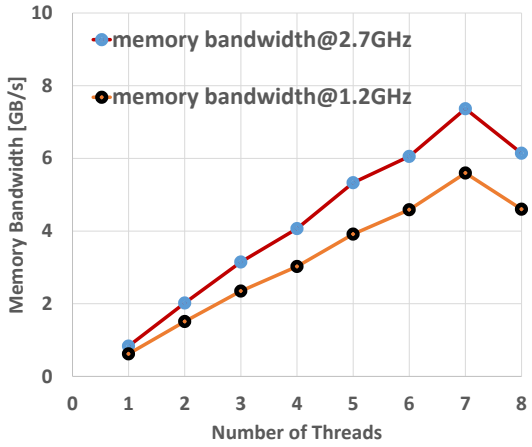
(d) Jacobi Benchmark Code

Figure 4.3.: (a,b) The on-chip power dissipation is a quadratic polynomial in the clock frequency at fixed number of threads (see Chapter 5 for a detailed analysis). (c,d) The on-chip power has a linear dependence on the number of active cores at fix clock speed of the "phinally" machine

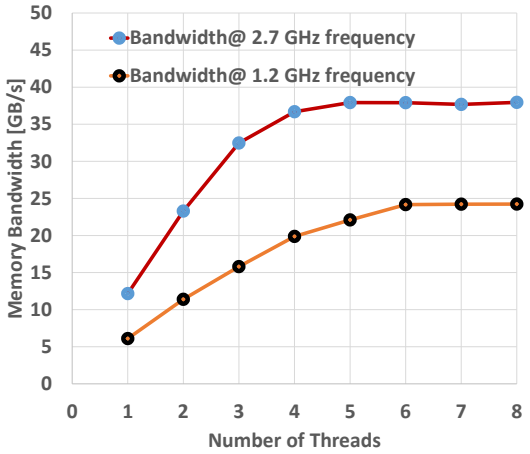
In Fig. 4.4(e), a drop in DRAM power at 8 cores is observed instead of expected perfect linear trend of a purely compute-bound application. This can be justified by fact that depending on different number of threads and problem size per thread, it may choose different blocking factor while switching from one thread count to other. This implies a decrease in pressure on memory interface. Also, as DGEMM code is far away from bottleneck (8 GB/s compare to 42 GB/s for "phinally" machine), this fluctuation of 2 GB/s bandwidth is really negligible. The power dissipation of installed DRAM is in the range of 6 W to 15 W and between 15 W to 38 W per socket with variant number of cores utilized and frequency configurations for DGEMM and jacobi workloads, respectively.

Variation effect on Energy consumption

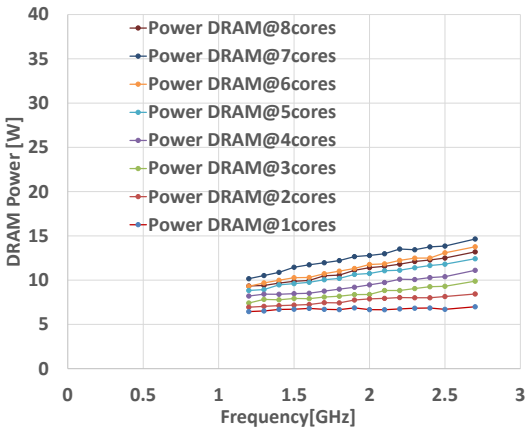
It is already known from the previous work that for a minimum on-chip energy to solution, a system should tune at optimum frequency with maximum number of available cores for core-bound cases and is near saturation point with lower frequency for applications limited by some bottleneck [10].



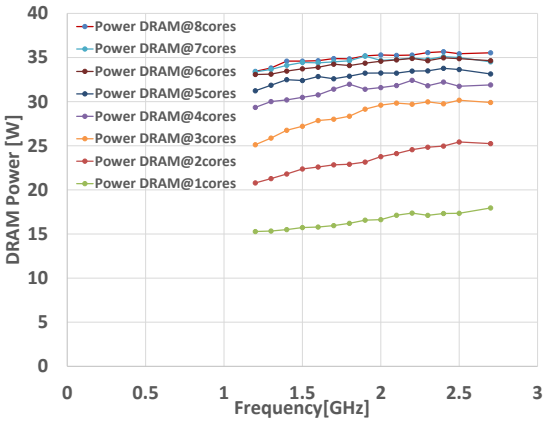
(a) DGEMM Benchmark Code



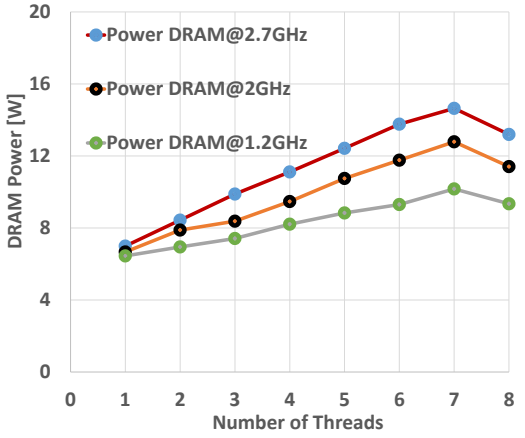
(b) Jacobi Benchmark Code



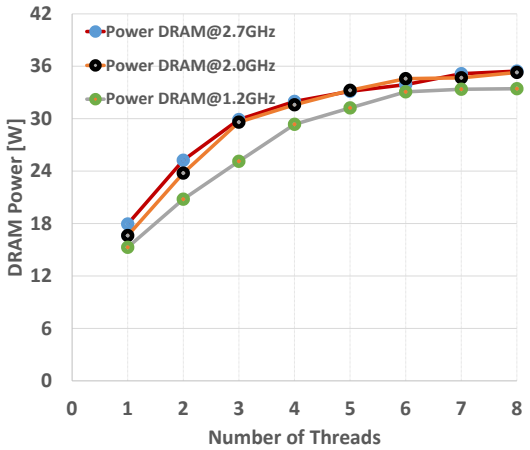
(c) DGEMM Benchmark Code



(d) Jacobi Benchmark Code



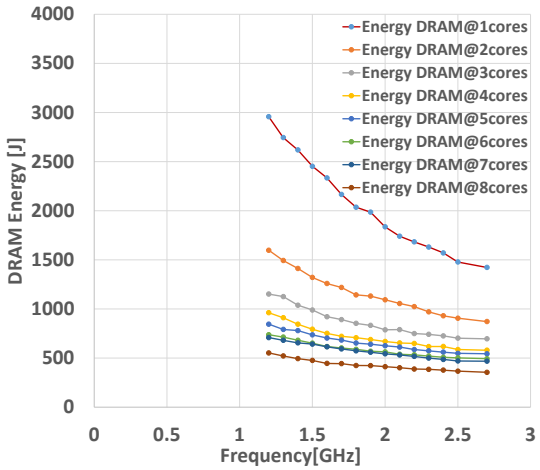
(e) DGEMM Benchmark Code



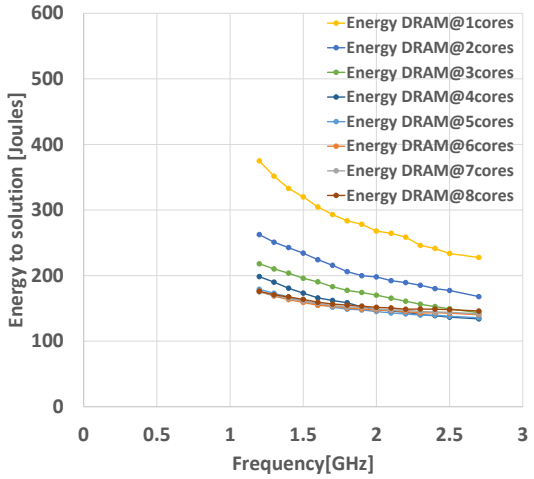
(f) Jacobi Benchmark Code

Figure 4.4.: A linear dependence of Memory bandwidth and dynamic DRAM power dissipation on the number of active cores and clock speed till saturation point on "phinally" system

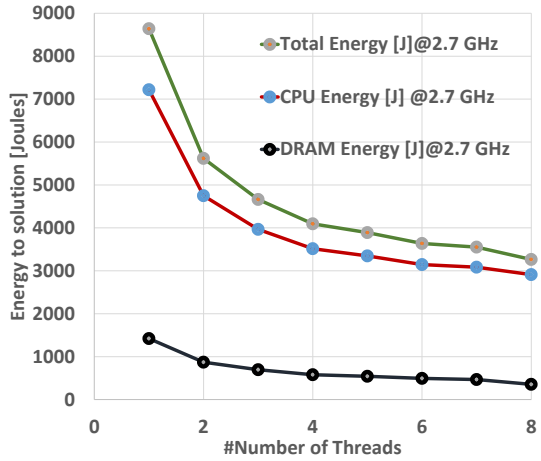
Measurements for DRAM energy demonstrate that it correlates with performance and the minimum DRAM energy to solution is obtained at the maximum value of performance. Hence, the optimal value of DRAM energy to solution of a system is achieved at the highest clock frequency and maximum number of available cores until it hits a bottleneck and remains constant



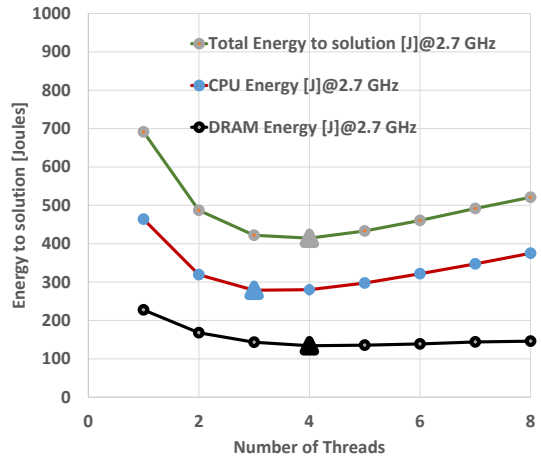
(a) DGEMM Benchmark Code



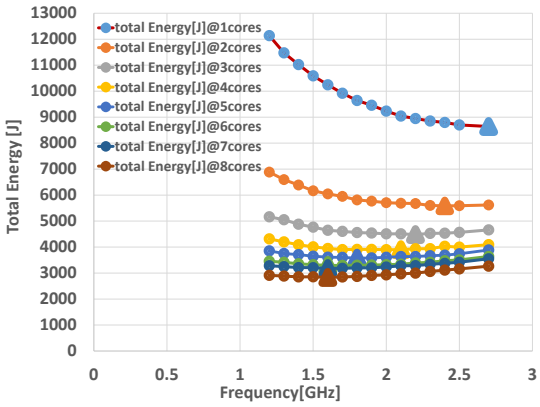
(b) Jacobi Benchmark Code



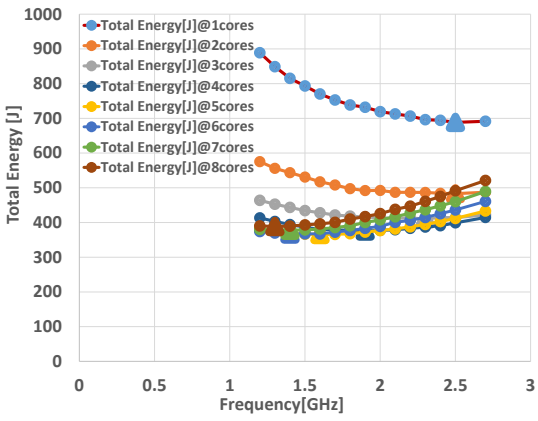
(c) DGEMM Benchmark Code



(d) Jacobi Benchmark Code



(e) DGEMM Benchmark Code

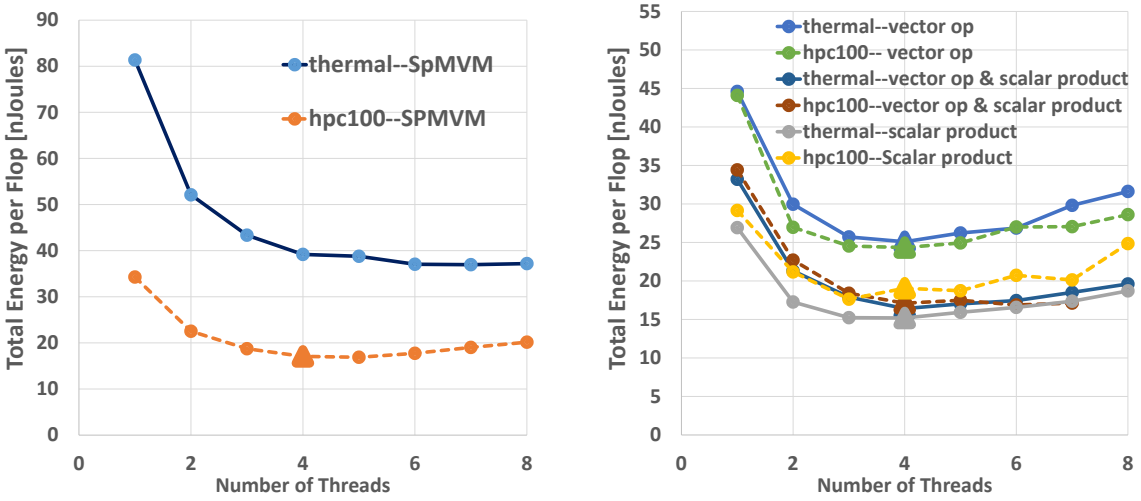


(f) Jacobi Benchmark Code

Figure 4.5.: The energy consumption with variation of cores utilized and clock speed on "phinally" machine

beyond saturation (see Fig. 4.5).

The figure 4.5(c) illustrates that a drop in energy consumption measurements at 8 cores is directly related to the drop in memory BW pressure as discussed in Sect. 4.4. In Fig. 4.5(d), while looking at energy measurements it is detected that with 3 cores Jacobi benchmark was



(a) spMVM in CG method

(b) Scalar product and vector operations in CG

Figure 4.6.: The energy consumed by different steps of CG algorithm with variation of cores utilized on "phinally" system

already close enough to saturation; however, switching from 3 cores to 4 cores makes the Jacobi application a still scalable with a bit more performance and implies a little bit save in DRAM energy. Practically, its somehow difficult to get the vicinity of the saturation point. Moreover, for total energy to solution results, it can be seen that beyond saturation the rising effect of the chip energy is damped a bit by constant DRAM energy. Hence, in order to consider a total energy to solution of a machine, DRAM energy to solution can't be neglected especially for an architecture with large installed memory.

Finally we take a look at the spMVM and BLAS-1 operations in the CG algorithm (see Sect. 4.3) from an energy consumption point of view. Figure 4.6 illustrates a comparison of overall energy consumption per flop for one iteration of disorganized "Thermal2" matrix and well-organized "Hpccg-100" matrix. Comparison exhibits that the well-defined "Hpccg-100" matrix shows a better performance and saves energy. In addition, energy consumption measurements detect an optimized overall energy consumption at 4 cores for each step of conjugate gradient algorithm.

Chapter 5

An Analysis of Analytical Models and Validation

Most of the previous works on power modelling and estimation have focused primarily on multi-core CPU dynamic power and ignore DRAM power consumption. This chapter describes the refinement of the previously defined simple CPU power model (in Section 2.4) and introduces a DRAM power model as well while considering the interesting features of both multi-core CPU and DRAM power dissipation by studying selective benchmark codes discussed in previous chapter. Moreover, the predicted model are validated against previously shown experimental measurements using different workloads.

5.1. CPU Power Model Refinement

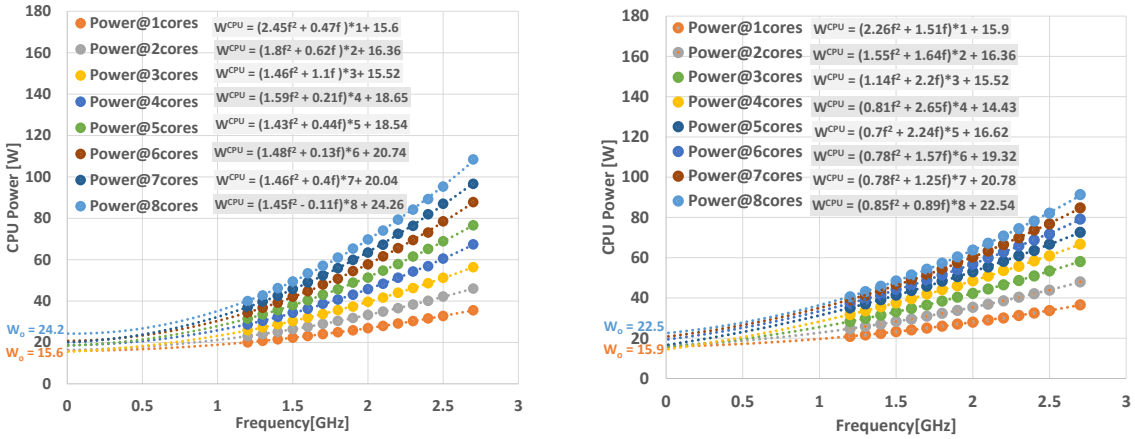
Previous multi-core chip power model by Hager el al. (see section 2.4) is

$$W(f,n) = \quad W_0 + W_1 nf + W_2 nf^2 = W_0 + w_1 f + w_2 f^2 \tag{5.1}$$

where $w_1 = W_1 n$ is measured in W/GHz and $w_2 = W_2 n$ is measured in W/GHz^2. Parameters W_i, i = 0,1,2, are determined by curve fitting using Levenberg-Marquardt method (discussed in Section 3.5.3).

The Figure 5.1 detect that the on-chip power estimation for the "phinally" test system using this model was fairly accurate. For example, for fixing W_i parameters, two benchmarks DGEMM and Jacobi were considered. For both applications, the linear factor w_1 is very small compare to the quadratic factor w_2 and constant baseline power W_0 as expected and both the W_1 and W_2 factors vary with core count. Moreover, when the Jacobi application hits the memory bandwidth bottleneck, cores begin to be starved for data and the observed slow increase in CPU power is reflected by higher value of the linear factor w_1 compare to the quadratic factor w_2. However, preliminary results for the CPU power dissipation indicate that the assumption of this model about extrapolated baseline power consumption W_0 (i.e, the whole chip consumes same W_0 independent of number of the active cores) seems not true, since it varies with switching on different number of cores. Baseline power consumption W_0 is still independent of type of running code but varies in the range from 15 W to 24 W depending on number of cores utilized for the "phinally" test system.

The figure 5.2 shows a chip with some cores over it. It illustrates the baseline power modification concept that instead of fixed baseline power, some baseline power goes away when a



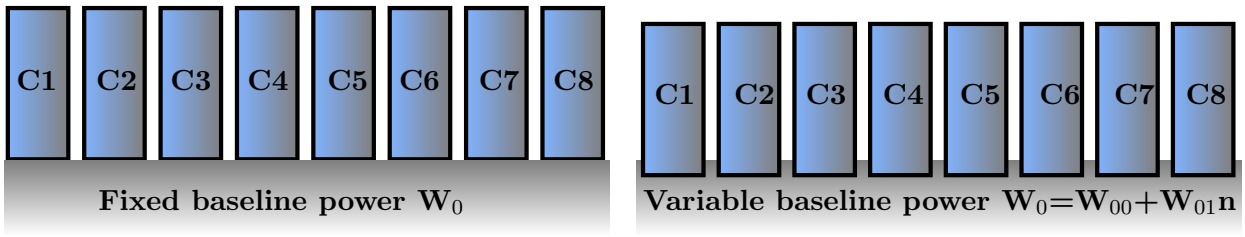
(a) DGEMM Benchmark Code **(b)** Jacobi Benchmark Code

Figure 5.1.: A comparison of measured CPU power dissipation data points (circles) with the CPU power model (5.1) (dotted curves) on "phinally" platform

certain active core is switched off. Considering this concept, some refinements have been done in multi-core chip power modelling with making baseline power W_0 as a combination of w_{00} and w_{01}:

$$W^{CPU}(f,n) = \quad W_{00} + (W_{01} + W_1 f + W_2 f^2)n = \quad W_{00} + w_{01} + w_1 f + w_2 f^2, \tag{5.2}$$

with $W_0(n) = W_{00} + w_{01}$, where W_{00} is the part of the baseline power which is really constant irrespective of number of cores utilized and $w_{01} = W_{01}n$ is the variable part of the baseline power which varies with switching on different number of cores and has a smaller contribution of the variable part W_{01} compare to the fixed part W_{00}. For "phinally" system, the absolute real baseline power W_{00} is around 14 W which is independent of anything and extra W_{01} of about 1.2 W per core is added when switch on a core.



(a) Previous Baseline Power Model (b) Modified Baseline Power Model

Figure 5.2.: The baseline power model for a single-socket 8 cores system node

The modified CPU power dissipation model for optimal frequency f_{opt} of scalable applications is compared against measurement curves. This model fairly describes the actual optimal clock speed with a difference of about 0.1 GHz at some points as shown in Fig. 5.3. Also the optimal clock speed slows down when switch on different number of cores which is reflected by much smaller contribution of variable baseline power W_{01} against quadratic factor W_2.

$$f_{opt}(n) = \sqrt{\frac{W_{00} + W_{01}n}{W_2 n}} = \sqrt{\frac{W_0(n)}{w_2}} \tag{5.3}$$

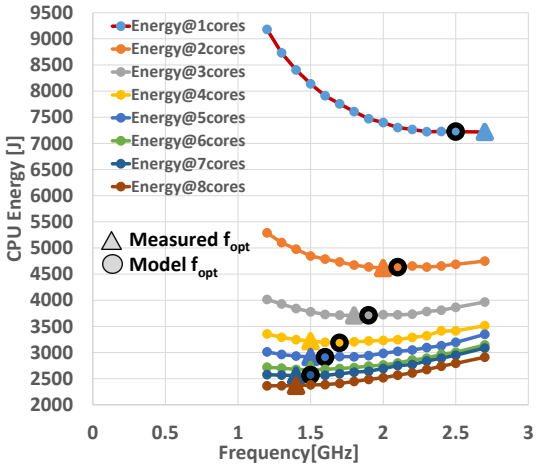


Figure 5.3.: A comparison of measured optimal frequency with modified modelled optimal frequency f_{opt} of the dynamic CPU power dissipation model for DGEMM benchmark code on "phinally" system

5.2. DRAM Power dissipation Model

With the widespread use of multi-threaded, multi-core processors, the rapid increase in demands on installed memory bandwidth and capacity is fulfilled by replacing conventional memory designs with new proposed designs. However, memory accesses become slower with respect to the processor ("memory wall" [25]) and consume more power with increasing memory system and complexity of applications. Thus, the focus of memory performance and power consumption has become increasingly important in the overall system power profile.

Memory system behavior and architecture of each type of DRAM system (i.e., SDRAM, DDR, and more) is sensitive to supportable bandwidths and topologies (i.e., number of DIMMs, number of channels), which in turn affects performance. However, trend for memory system performance is difficult to discern as it is affected by interaction of many parameters including DRAM timing parameters, memory system topologies and memory controller policies [26]. Moreover, a comparison of the memory system architectures becomes non-trivial when the dimensions of the power consumption and the manufacturing cost are added.

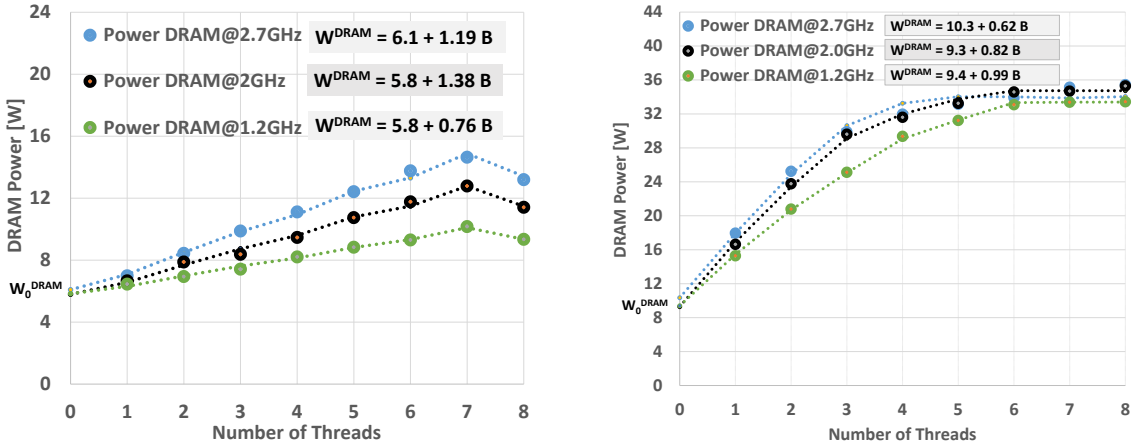
Starting with the Intel Sandy Bridge processor, it has become possible to directly measure the power dissipation of dynamic random access memory and correlate this data with the performance properties of the running code depending on the workload (in Section 4.4). From analysis of preliminary results for the power dissipation of installed DRAM on benchmark programs, a simple, phenomenological DRAM power model is derived:

$$W^{DRAM}(B) = \quad W_0^{DRAM} + wB \tag{5.4}$$

where, memory bandwidth $B = min(n{*}f/f_0{*}B_0, b_s)$ is the limiting factor measured in GBytes/s, w is a constant, in $\frac{Watts}{GBytes/s}$, which tells that how much power is needed for each GB/s drawn from memory. Further, W_0^{DRAM} is the background power that a DRAM consumes all the time with or without operations. Furthermore, the actual parameters (slope w, base power W_0^{DRAM}) depend on the particular type and number of DRAM DIMMs, and they are determined by curve fitting method.

This dynamic random access power model takes the memory bandwidth as input parameter and predicts the power burnt in DRAM for different workloads with variant multi-core density

and frequency configurations. Further, this model suggests a DRAM power proportionality with memory bandwidth and certain background power. To validate the DRAM power dissipation model, measurement curves are compared against modelled curves. It was observed that the background power W_0^{DRAM} is smaller for DGEMM code instead of being constant. This effect might be explained by its smaller bandwidth utilization compared to Jacobi case that causes the DRAM power to go to a power saving state. However, it does not happen with Jacobi case because it constantly stream data and pre-fetchers work perfectly. Hence, the DRAM power model describes nicely the actual measured memory power consumption (see Fig. 5.4).



(a) DGEMM Benchmark Code **(b)** Jacobi Benchmark Code

Figure 5.4.: A comparison of measured power dissipation data points of memory (circles) with the DRAM power dissipation model (5.4) (dotted lines) on "phinally" platform

5.3. Multi-core Total System Power dissipation Model

This section provides a complete model for estimation of whole system dynamic power consumption of an installed system. This refined overall system power consumption model also considers the modelling for dynamic power consumption of memory module in addition to CPU power modelling, since DRAM power can't be neglected especially for an architecture with a lot of memory installed. The overall power dissipation model is

$$\begin{aligned} W^{Total}(f,n,B) &= W^{CPU}(f,n) + W^{DRAM}(B) \\ &= W_{00} + (W_{01} + W_1 f + W_2 f^2)n + W_0^{DRAM} + wB \end{aligned} \tag{5.5}$$

5.4. Multi-core Total System Energy to solution Model

At the heart of many approaches towards energy efficient computing lie energy models that allow extrapolating the future energy consumption of a system. The model for estimating the overall energy to solution of whole system during a certain period of time is obtained as:

$$E^{Total}(f,n,B,T) = W^{Total}(f,n,B,P) * T \tag{5.6}$$

where W^{Total} is the overall dynamic power dissipation, T is the total execution time of that program and $T = \frac{1}{P}$ is an assumption that is valid if we always solve the same problem (i.e., the "work" can be normalized to one). Therefore, the energy consumption at the granularity of

an application is also calculated by concurrently estimating the overall system dynamic power to performance ratio:

$$\begin{aligned} E^{Total}(f,n,B,P) &= E^{CPU}(f,n,P) + E^{DRAM}(B,P) \\ &= \frac{W_{00} + (W_{01} + W_1 f + W_2 f^2)n + W_0^{DRAM} + wB}{min((1+\frac{\Delta f}{f_0})nP_0, P_{max})} \end{aligned} \tag{5.7}$$

It should be noted here that the performance model makes a decisive simplification here: We assume that the performance is linear in the clock speed until a bottleneck is hit. This is not strictly true for strongly memory-bound loops, but the error is small.

This energy to solution model considers both the energy consumed by the whole CPU and DRAM module. It is used to account the energy usage to the respective system for allowing an application behaviour to remain within externally specified energy constraints. Many interesting conclusions for system design can be drawn from this model with considering typical requirements of a computing centre. This energy model suggests that in order to save energy, a machine should run at optimal frequency with maximum core counts for compute-bound codes, and at lower frequency with core count near saturation for memory-bound codes [10]. Moreover, if we look at consequences of the modification of the power model (i.e., addition of W_{01} and DRAM power W^{DRAM}) for these conclusions then there are no changes for scalable codes, because the DRAM power is then a contribution to W_{00} and W_1 (since B is linear in n and f). However, for saturating codes, $B = min(n * f/f_0 * B_0, b_s)$, although the numerator in modified energy model picks up a different characteristic from before but still it does not make a big difference for these conclusions.

Chapter 6

Consequences for Code execution

To better understand how to achieve an energy-efficient execution of parallel programs, one needs to see where the power is drawn, and to identify possible ways to increase the energy efficiency of the system. Together with the previously described models and results, this section explains many peculiarities in the performance and power behaviour of multi-core processors, and derives guidelines about running scientific computing workload in best possible way. Later, a power and energy comparison of the different architectures available at RRZE computing centre is introduced in Section 6.4.

6.1. Energy delay product and its generalization

To derive a general necessary condition under a time-energy trade-off for improving the energy efficiency, a more appropriate metrics such as "Energy delay product (EDP)" and its generalization metrics are targeted which give more information than pure "Energy to solution" metric:

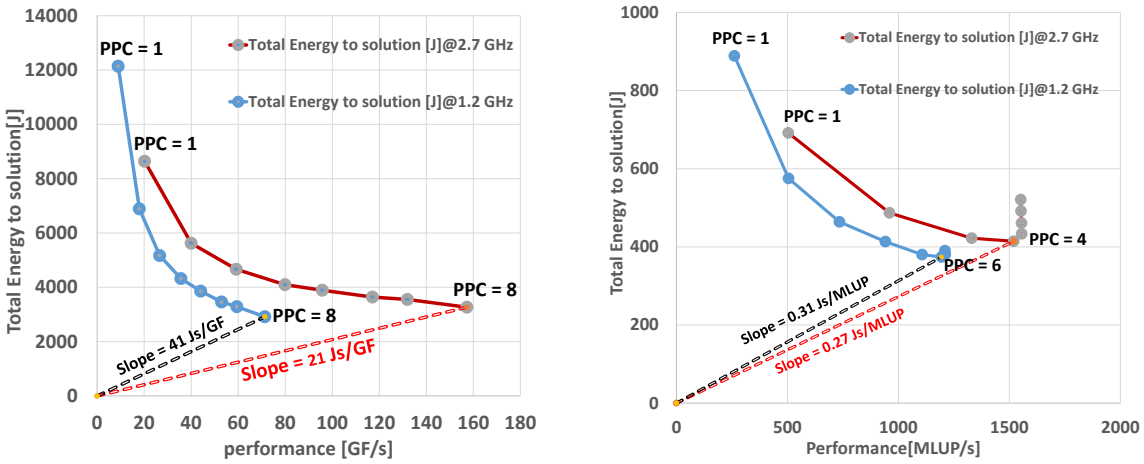
$$E^{Total}(f, n, B, T) * T^i = \frac{E^{Total}}{P^i} = \frac{W^{Total}}{P^{2i}}, \tag{6.1}$$

where i is a small number (i.e., i= 1,2,3), T is the required total execution time for running an application and P $= \frac{1}{T}$ is an assumption that is valid if the "work" can be normalized to one, i.e., if we always solve the same problem. In particular, that rules out that we look at these metrics in a weak scaling scenario. The energy delay product EDP and its generalization ED^2P and ED^3P metrics are measured in Js, Js^2 and Js^3, respectively.

The energy delay product determines and quantifies that the improvements in energy-efficiency may or may not require a slowdown. In other words, it shows whether the loss in performance is outweighed by the gain in energy efficiency or not. Many approaches towards energy-efficient system select configurations to avoid performance loss whereas some allow for a certain (constrained) performance loss while saving energy.

There are general scenarios to expect an time-energy trade-off at the granularity of an algorithmic properties or a micro-architecture of the system. One relevant approach for considerations of the running code is to trade more compute operations for less data transfers. On the other hand, from an architecture perspective, when we play around frequency then there happens some contradiction between maximum performance and minimum energy consumption. For instance, the reduced frequency setting saves energy but at the same time we have

to compromise performance, due to the quadratic relationship between the frequency and the CPU power (see Fig. 6.1).



(a) DGEMM Benchmark Code **(b)** Jacobi Benchmark Code

Figure 6.1.: A "Z-Plot" of the energy to solution versus performance for available number of cores utilized with optimal value of "Energy delay product" on "phinally" machine

For studying energy to solution and EDP, it is quite useful to plot the energy consumption and the performance in a "Z-Plot", which illustrates the energy to solution versus performance for different core counts. In a Z-plot, all points with a constant energy-delay product lie on a straight line through the origin. In fact, the slope of this line is equal to the EDP if P=1/T and EDP unit varies with the unit of performance due to our convention of P=1/T.

The "Z-plot" is useful for quantifying the amount of the energy reduction at the cost of performance loss. It suggests that one would like to operate a code as far to the right and bottom as possible in the "Z-plot" to save energy. It illustrates that in this "Z-Plot" how the basic characteristics of scalable (DGEMM-like) and saturating (Jacobi-like) codes look like, i.e., the performance improvement with the energy reduction continue to grow with the increasing number of cores until it hits a bottleneck; whereas, after saturation the energy consumption starts to increase with the performance remains constant. Furthermore, the "Z-plot" explains the general behaviour with the clock speed changes, i.e., the clock speed reduction causes a large growth in performance loss for scalable codes comparable to saturating codes where we have a comparatively smaller performance reduction.

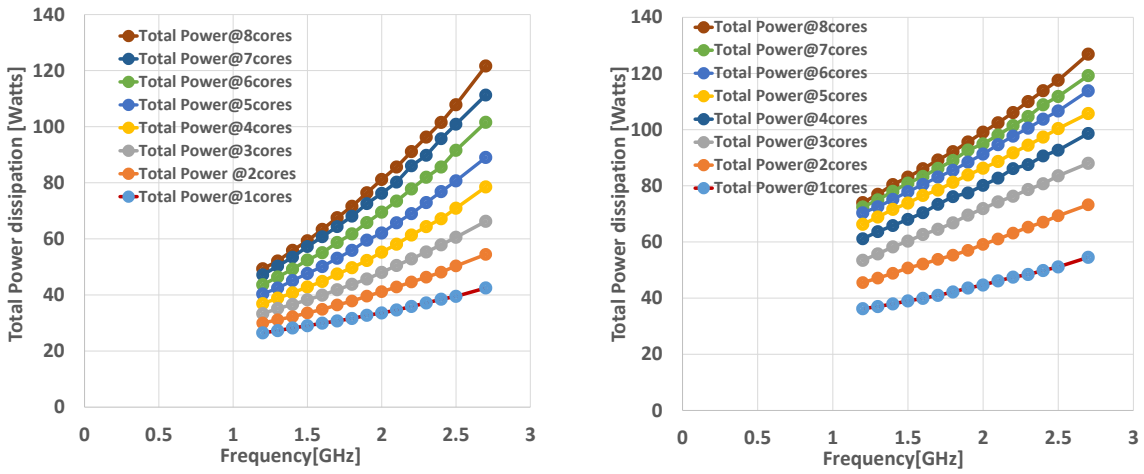
The "energy efficiency" of a system is a "low EDP". The large performance sacrifice was observed at lower frequency setting for scalable applications (i.e., almost double slope of the EDP for DGEMM code). Therefore, for energy efficient execution of a scalable application, the energy model 5.4 suggests that it is highly recommendable to run such applications at optimal or higher frequencies with maximum number of available cores with or without sacrificing performance, respectively. However, for energy efficient execution of a saturating application, the performance loss near saturation point for the lower frequencies is not so high compare to a core-bound case. Thus, this is totally dependent on desirable goal that either it allow a performance loss or not. For instance, to achieve the minimum overall energy consumption it is suggested to run Jacobi smoother at 4 cores with 2.7 GHz without compromising time to solution or to run it at 6 cores and 1.2 GHz with a maximum increase in time to solution, respectively. However, the EDP is larger in the latter case.

6.2. Power Capping

Today power dissipation, being a limiting factor, has emerged at the forefront of challenges facing the microprocessor designer. In the past, the cooling infrastructures were designed for the worst situation of a microprocessor in which the processor is constantly operating at its theoretical maximum power and thus dissipates a maximum of heat. However, such components are costly and cannot be expected to scale to the higher power levels as transistor dimensions shrink. Although, most ordinary tasks do not cause the processor to consume this maximum power but when they do so the alternative is to choose a more moderate thermal design power.

Nowadays, the goal of some scientific computing centre environments is to keep the overall power dissipation below some threshold (i.e., $W^{Total} \leq W_{max}$), which is called power capping. This power capping feature does not work with energy to solution or energy delay product metrics, in fact it strictly works with power dissipation of a system. Therefore, it is important to look at different general approaches to run an application to be cooler but with the same performance.

The properties of an algorithm can help to inform power management. For instance, if given allowed power dissipation of Sandy Bridge EP processor in "phinally" system is 120 W, then overall power dissipation is higher for both scalable DGEMM and saturating Jacobi cases as shown in Figure 6.2. However, for saturating applications, the goal of lower power dissipation is easily achievable by running this application with smallest necessary number of cores near saturation point without limiting performance (e.g., at run jacobi code at 5 cores in Figure 6.2(b)). Although, for scalable applications, this goal of power capping requirement is more complex and difficult to achieve. A processor must be clock down if it becomes too hot after executing scalable tasks which exceed this power constraint. Hence, frequency reduction degrades the system's performance with less science achieved and should only be applied if really necessary. For example, in Figure 6.2(a), DGEMM code has the whole power dissipation of 123 W at maximum 8 number of cores of "phinally" system with 2.7 GHz base clock speed. Suppose this power dissipation is above threshold value then one has to sacrifice time to solution by clock speed slowdown to achieve the power capping goal.

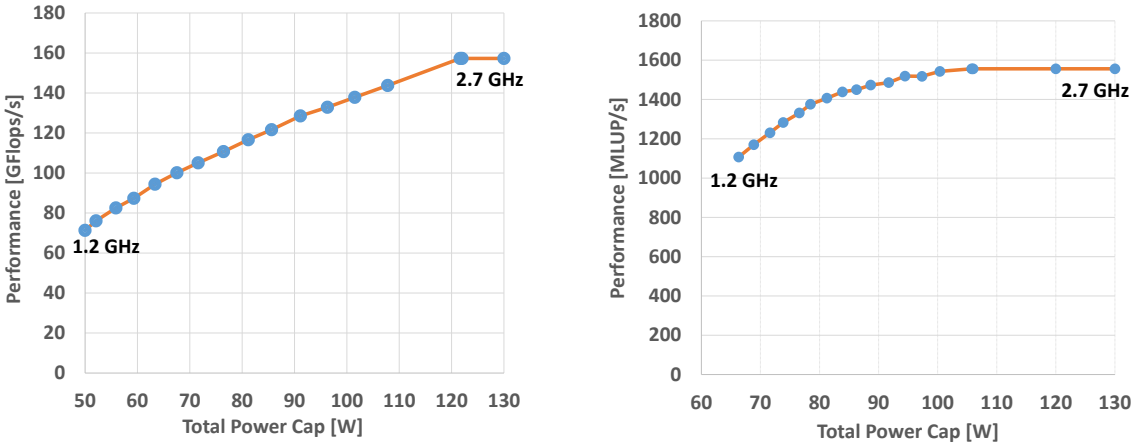


(a) DGEMM Benchmark Code **(b)** Jacobi Benchmark Code

Figure 6.2.: Overall power dissipation of a Sandy Bridge EP processor in "phinally" system.

Figure 6.3 shows a graph with "foced performance loss versus overall (CPU + DRAM) power

cap", which quantifies how much performance needs to sacrifice if power capping is really present. This curve is largely horizontal for saturating Jacobi but sloped for scalable DGEMM as expected. It illustrates the suitability of Jacobi code for running under power capping. If there is a power cap, we may be able to run Jacobi with no noticeable performance loss but without hitting the power cap, just by setting a low frequency and adjusting the core count.



(a) DGEMM Benchmark at 8 cores

(b) Jacobi Benchmark at 5 cores

Figure 6.3.: Overall power dissipation vs. power cap of a Sandy Bridge EP processor in "phinally" system.

6.3. Trading performance for energy

This section elaborates general approach to treat problem slowdown under a time-energy trade-off. For instance, some goals like power capping of computing centres result in the performance loss by reducing the clock speed of scalable codes as discussed in previous Section 6.2. This is an interesting dimension that things get more complicated with sacrificing the performance.

There are two general approaches to build an energy-efficient system design compared to their baseline power W_0. First approach is to build a very cool systems like BlueGenes with very small baseline power W_0. These systems are typically slow because of low clock speed but they are very parallel with hundreds of thousands of processors. On the other hand, second approach is to build very fast and hot processors with large value of baseline power W_0 (e.g., one processor with 1000 watts). This approach is very interesting but practically no one build such systems as technology challenges are very hard and such a large watts of space is not feasible. These two corner cases (i.e., many cold CPU with low clock speed and few hot CPU with high clock speed) are examined with energy cost model to trade some performance for saving the energy consumption.

Suppose if the performance reduction is accepted for considered energy cost model system in Section 2.2 then it means that this machine always have less science for its six years of operational runtime. However, this saved money can be used in energy to buy a larger system for compensate performance loss. We can assume the point of view that a certain system is running at the optimal clock speed f_{opt} and then adjust the size of the system to compensate for the performance loss. Hence, now this considered system will be cooler but probably larger to get same science over six years. In addition, the potential of larger machines towards energy saving needs to be targeted that either it costs more or less.

If we neglect the small linear part in the power model 5.2, then the ratio of power dissipation between the optimized and the base clock frequencies is smaller than one for equal sized systems:

$$\frac{W(f_{opt},n)}{W(f_o,n)} = \frac{W_0(n)+W_2 n f_{opt}^2}{W_0(n)+W_2 n f_o^2} = \frac{2W_0(n)}{W_0(n)+W_2 n f_o^2} \tag{6.2}$$

However, if we adjust the size of the optimized system by a factor $\frac{f_o}{f_{opt}}$ that reflects the chip performance ratio (and assume perfect scaling for applications), so the resultant dimensionless ratio R quantifies the energy saving potential of the large machine to compensate for the loss in performance. The R metric can explore the design space of possible parallel machines with different values for W_0, W_2, and n.

$$R = \frac{W(f_{opt},n)}{W(f_o,n)}\frac{f_o}{f_{opt}} = \frac{2f_o\sqrt{W_0(n)W_2 n}}{W_0(n)+W_2 n f_o^2} \tag{6.3}$$

$$R_{cool} \approx \frac{2}{f_o}\sqrt{\frac{W_0(n)}{W_2 n}} = \frac{2f_{opt}}{f_o}; \qquad W_0 \ll W_2 f_o^2 n \tag{6.4}$$

$$R_{hot} \approx \frac{2f_o\sqrt{W_2 n}}{\sqrt{W_0(n)}} = \frac{2f_o}{f_{opt}}; \qquad W_0 \gg W_2 f_o^2 n \tag{6.5}$$

It is straightforward to show that the energy saving potential matrix $R = 1$ for $W_0 = W_2 n f_o^2$, and $R < 1$ otherwise. A value of $R = 1$ marks the inflection point where it is not possible to save energy by trading clock speed for machine size. In the limit of very small baseline power $W_0 \ll W_2 f_o^2 n$, a large number of cores per chip thus favors large, cool systems. On the other hand, energy can also be saved with a very high clock frequency for "hot" machines where $W_0 \gg W_2 f_o^2 n$.

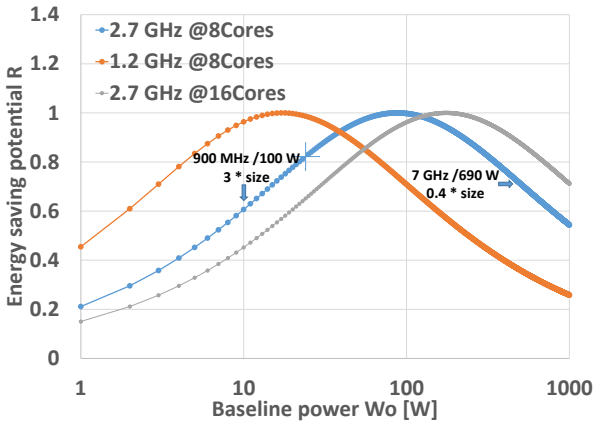


Figure 6.4.: Energy saving potential R vs. baseline power $W_0(n)$ for running a system at the optimal clock speed f_{opt} and 1.5 W dynamic power parameter W_2 with a scalable code and adjusting the system size for constant aggregate performance.

If we plot $R(W_0)$, it is the combination $W_2 f_o^2 n$ which determines the shape of the curve, and especially the position of the inflection point R = 1 (see Fig. 6.4). At a given value of $W_2 f_o^2 n$, the region left of $R = 1$ is where a clock slowdown can save energy corresponding to larger cool machine, because the baseline power is small. The region to the right of R = 1 is where "clock

race to idle" applies , see Section 2.4 (and a correspondingly smaller machine). Here the chip is so hot that it is beneficial to run at very high clock speed to "get it over with".

Figure 6.4 shows three different scenarios by choosing a different number of cores and a different base clock speed f_o. Note that any change in W_2 can be mapped to a proportional change in n, so W_2 was fixed to 1.5 W/GHz^2, which is roughly the measured value for scalable codes with the Intel Sandy Bridge EP processors in the "phinally" system at RRZE. For the case $f_o = 2.7$ GHz at eight cores, the inflection point $R = 1$ point is at $W0 \approx 90$ W. All other parameters being equal, a small baseline power of $W_0 = 10$ W leads to a system that can be made 3 times larger, runs at $f = 900$ MHz, and dissipates about 20 W per chip (including the dynamic power). On the other hand, if $W_0 = 600W$ we get processors running at $f = 7$ GHz and 1200 W, but the system can be built at 40% of its original size. The plus point in Fig. 6.4 marks the position of "phinally", whose base frequency is 2.7 GHz at eight cores and an overall baseline power of $W_0 \approx 24$ W. Since this point is far from the maximum where $R = 1$, it is possible to save energy on this system by reducing the clock speed in favor of more hardware.

These considerations stretch the power model 5.2 very far, and it is not expected that the numbers derived above have any useful accuracy as these calculations assume a scalable code and really exact calculations for f_{opt}. However, the model is still interesting for qualitative design space exploration.

6.4. Results for different architectures

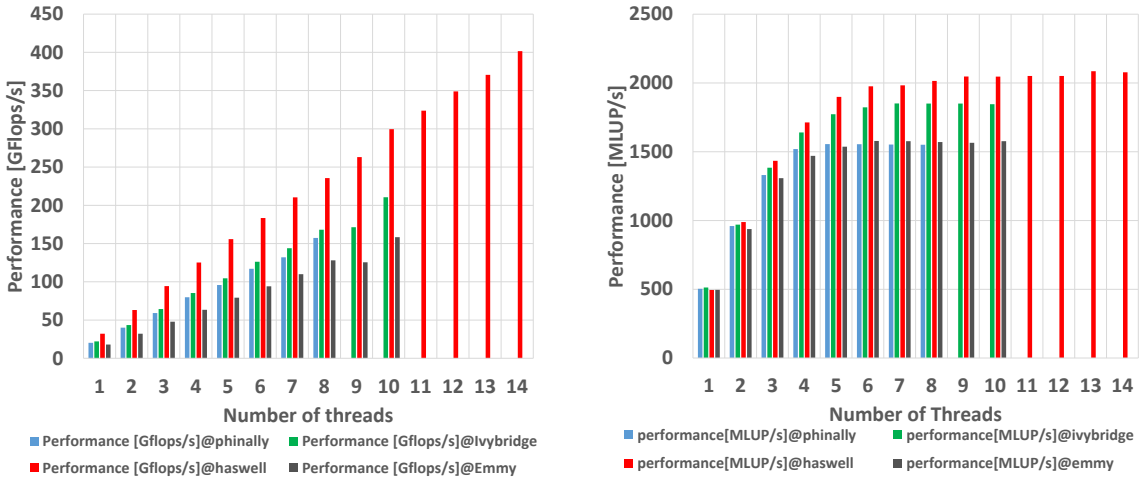
In this section we present the power dissipation and energy consumption behaviour for the test systems from the benchmark programs under different type of workloads. All architectures run Linux OS and use the intel compiler suites. The benchmarks are executed with the same input set size on all four different platforms discussed in Chapter 3 (i.e., phinally, emmy, ivybridge and haswell, 2.2 - 3 GHz base frequency) in order to have comparable numbers. Measurements of the performance, the power dissipation and the energy consumption of full socket are performed for all four platforms. Furthermore, from preliminary results of benchmark programs with vastly different requirements to the hardware, a simple comparable conclusion is derived about the characteristics for these platforms.

6.4.1. Comparison in terms of Performance

In terms of performance, the observed differences between the benchmark systems can be explained by their basic properties: peak performance and memory bandwidth (see Chapter 3). All systems show close to peak performance for DGEMM; the bandwidth-bound Jacobi code performs in accordance with the Roofline model (see Section 4.2). Figure 6.5 shows a summary of the in-socket performance scaling data for the two benchmark on all four systems.

6.4.2. Comparison in terms of Power

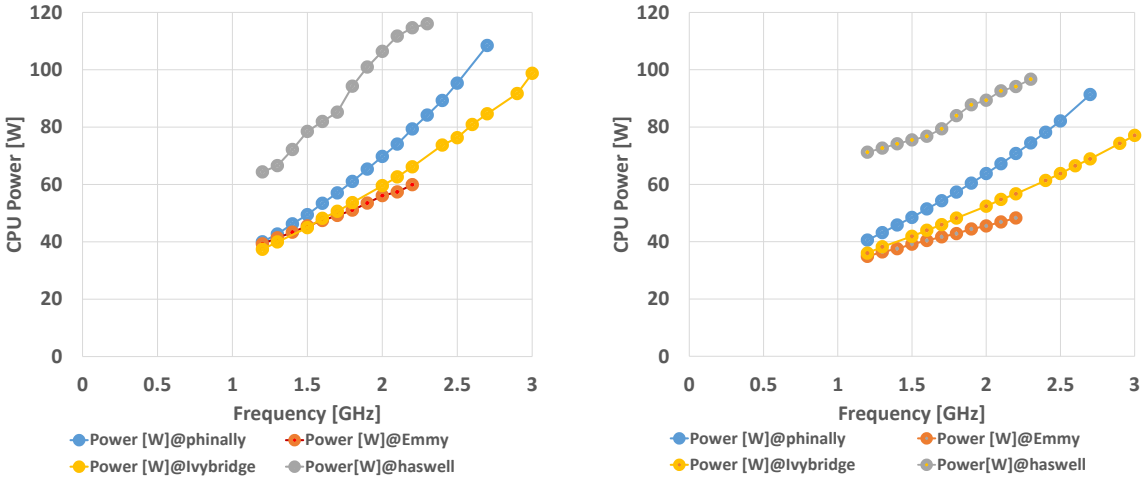
In the case of both core-bound and memory-bound workloads, it is observed that the haswell architecture in "hasep1" platform is significantly faster than others, but at the same time, is really a hot processor whereas other platform consumes less power (see Fig. 6.6). The increase in power occurs because of the overall rise in utilization and throughput of the processor, i.e., the haswell processor completes the same work in a shorter period of time. Preliminary



(a) DGEMM Benchmark Code **(b)** Jacobi Benchmark Code

Figure 6.5.: Performance of phinally, ivy bridge, haswell and emmy platforms at their respective base clock frequencies.

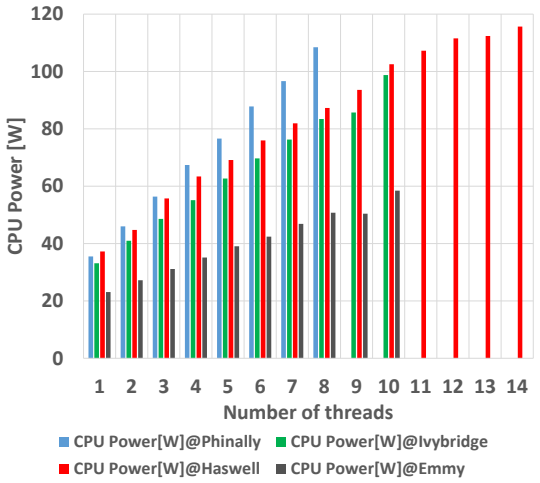
measurements for whole CPU also witness that DGEMM is not absolute hottest code and has more computational intensity compare to the hottest code.



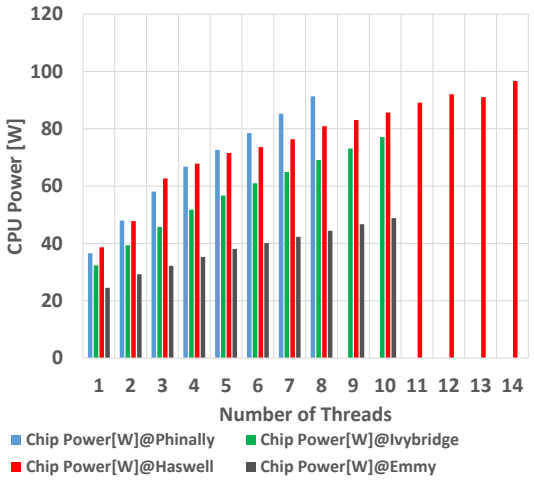
(a) DGEMM Benchmark Code **(b)** Jacobi Benchmark Code

Figure 6.6.: The on-chip power of phinally, ivy bridge, haswell and emmy platforms at their respective number of available cores.

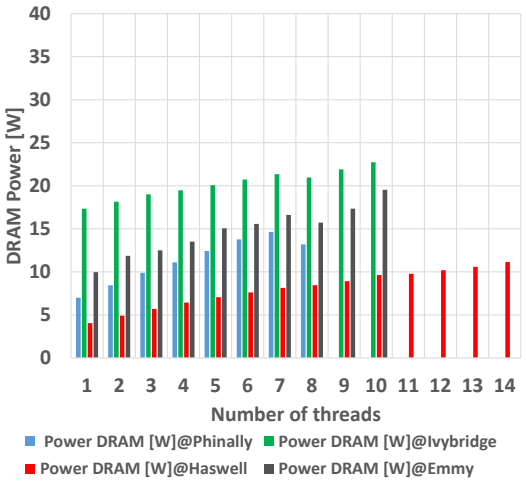
The power dissipation vs. clock speed data in Fig. 6.6 shows that the "phinally" and "ivyep1" systems are quadratic, the "emmy" platform is linear, and the "hasep1" machine is exceptional. When using the full chip, the exceptional behaviour of "hasep1" is justified by the fact that the chip tries to keep power below its Thermal Design Power (TDP) by all means [27]. However, one can be able to recover the smooth quadratic power frequency curve by using fewer cores, e.g., 10 or 12 cores instead of 14 cores. Furthermore, the chip power of Ivy bridge processor in the "emmy" platform illustrates a linear behaviour with the clock speed and implies a zero quadratic factor W_2 in multi-core CPU power model. In reality, the absolute values of the quadratic factor W_2 is very small compared to linear factor W_1, so that the linear behaviour dominates for the "emmy" cluster. This linear behaviour of chip power for "emmy" production cluster is justified



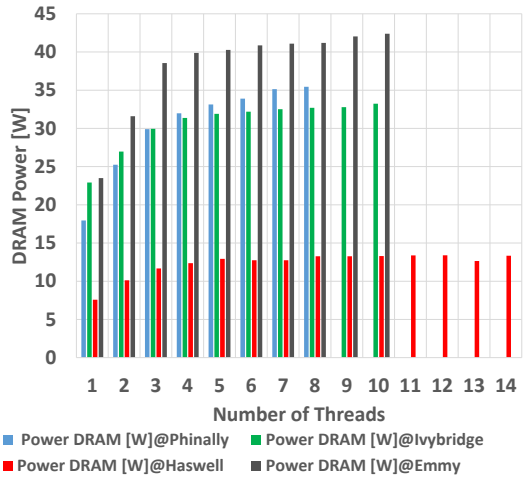
(a) DGEMM Benchmark Code



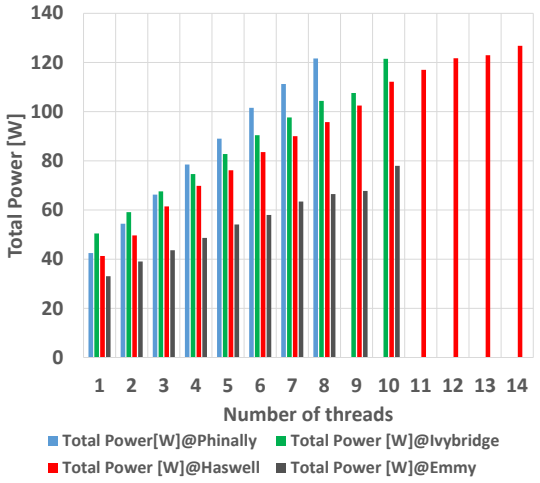
(b) Jacobi Benchmark Code



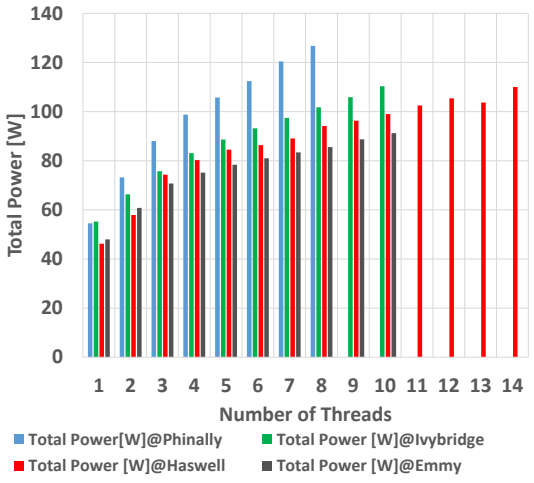
(c) DGEMM Benchmark Code



(d) Jacobi Benchmark Code



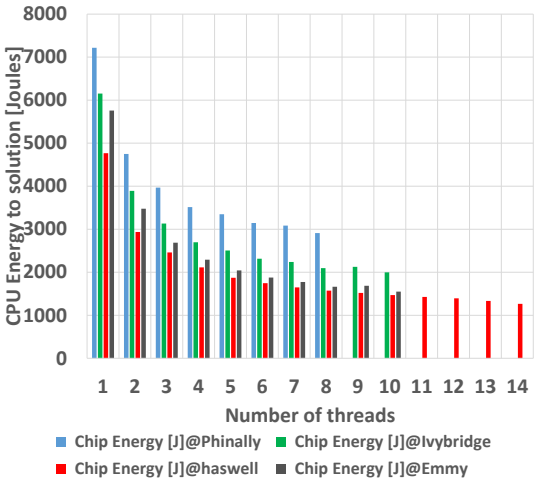
(e) DGEMM Benchmark Code



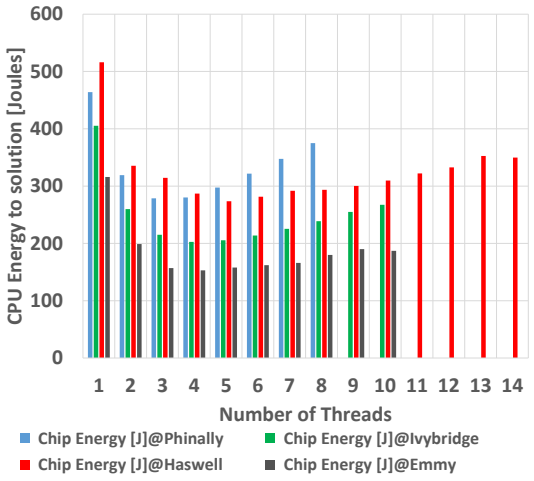
(f) Jacobi Benchmark Code

Figure 6.7.: The power dissipation of phinally, ivy bridge, haswell and emmy platforms at their respective base clock frequencies.

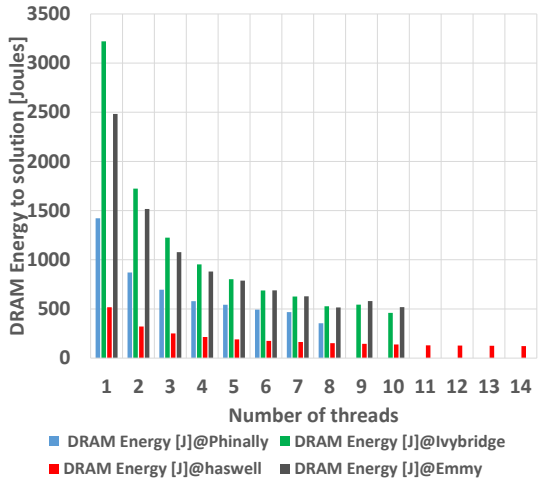
by its very low base clock speed. Due to the low base clock frequency of 2.2 GHz, there is presumably no room for voltage adjustments over the accessible frequency range, leading to a



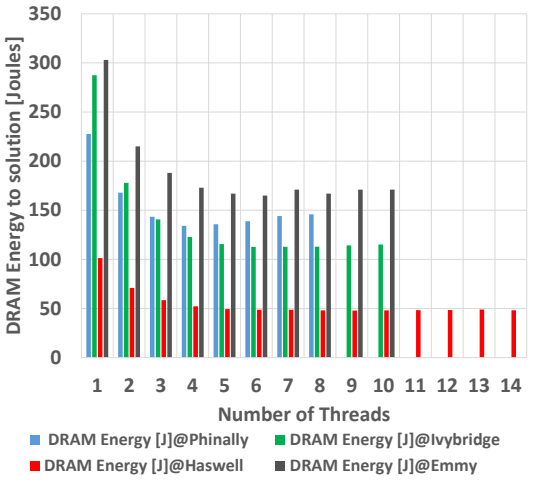
(a) DGEMM Benchmark Code



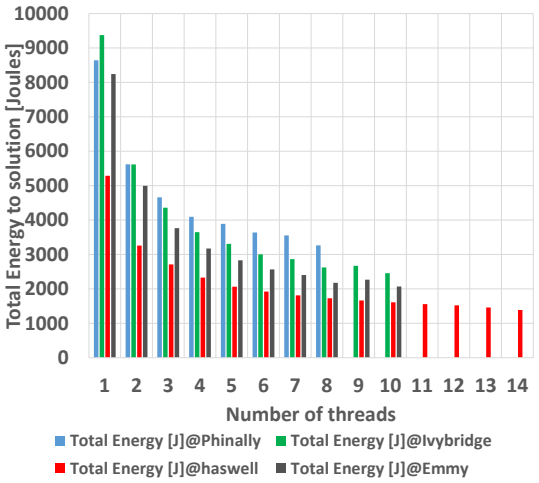
(b) Jacobi Benchmark Code



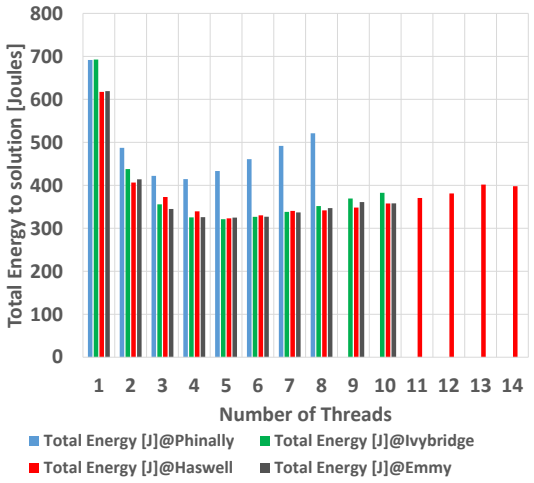
(c) DGEMM Benchmark Code



(d) Jacobi Benchmark Code



(e) DGEMM Benchmark Code



(f) Jacobi Benchmark Code

Figure 6.8.: The energy consumption of phinally, ivy bridge, haswell and emmy platforms at their respective base clock frequencies

linear power behaviour. Furthermore, the CPU power model suggests that the "emmy" system should always be run at its base clock speed to get an optimal frequency $f_{opt}(n) = \sqrt{\frac{W_{00}+W_{01}n}{W_{2}n}}$,

since W_2 is zero or very small.

When the power consumed by one socket is analysed, the dominant part of the total power is spent on the CPU cores, while a smaller (but still significant) fraction goes into the memory modules. Although by increasing number of cores (multi-core density) the overall power increases, the power consumed by shared components (such as memory) does not scale linearly with the number of cores. The clock speed of the DRAM effects the DRAM power, but there are other influence factors that can play a decisive role. The previous experiments suggest that the DRAM power is a linear function of the memory bandwidth delivered to the application, with a non-zero base power (see Section 4.4). The actual parameters (slope, base power) depend on the particular type and number of DRAM DIMMs, so we can only measure them. It is entirely possible that we could buy a node with the same performance characteristics of an emmy node but which consumes much less DRAM power because somebody has chosen a different DIMM manufacturer.

While looking at overall power dissipated by a core of haswell processor in "hasep1" machine, the fact that a haswell core dissipates less overall power comes from the particularly low DRAM power dissipation in that particular Haswell node (see Fig. 6.7). Intel has also equipped the haswell based processors with more power ratings and lower power modes so they can switch power modes faster than their predecessor.

6.4.3. Comparison in terms of Energy to solution

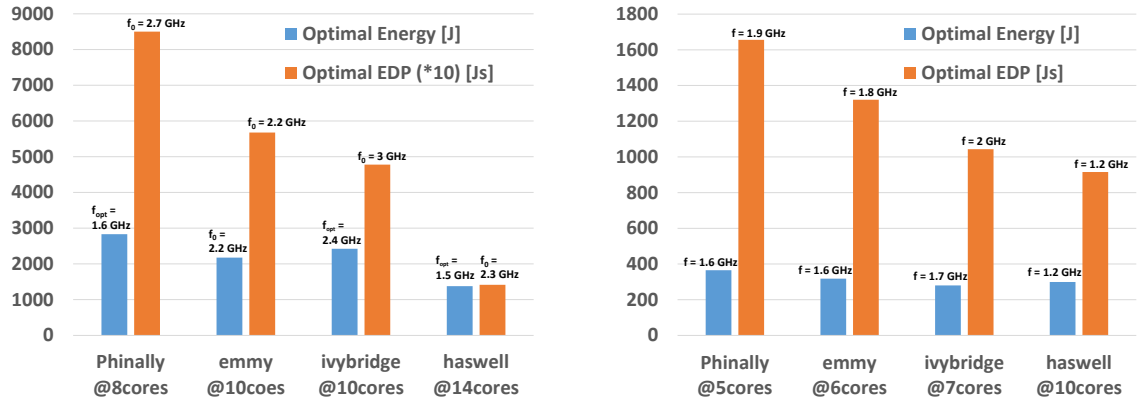
For scalable codes, like DGEMM, the implementation of SIMD floating-point unit comes at the cost of increasing power consumption but at the same time the instructions with boost floating-point performance result in improved energy efficiency. The previous studies elaborate that a "haswell" core is significantly faster than others (Sect. 6.4.1) and at the same time uses overall less power which this is a consequence of the low DRAM power of this particular system (Sect. 6.4.2). Thus, it results in a very smaller overall energy-to-solution per core and increases the overall energy-efficiency of the "haswell" system (see Fig. 6.8).

6.4.4. Overall comparison across architectures

To get maximum performance a system should be operated at the highest frequency, which causes highest power dissipation; however all aspects need to be analysed together for getting an energy-efficient system. An analysis has been performed to determine the desirable setting with variant cores and frequency to obtain an energy efficient execution of running applications. Two metrics are used for this analysis: minimum energy to solution and minimum energy delay product.

In figure 6.9, the blue bars represent the minimum energy to solution and the orange bars show the minimum value of EDP. It is extremely interesting to note that these comparison results collaborate and substantiate the findings from Section 6.1 for an energy-efficient system. It was stated in Section 6.1 that a system should run at maximum cores with optimal or base clock speed for core-bound case with or without sacrificing performance, respectively. Moreover, for saturating memory-bound case, a machine should run near saturation point at lower or higher clock speed with or without compromising performance, respectively.

The exceptional frequency setting with "emmy" can be observed with compute bound DGEMM application because of zero quadratic factor. However, it is not observed beyond saturation point for Jacobi case, since lower frequency is always better after a bottleneck is encountered and clock



(a) DGEMM Benchmark Code

(b) Jacobi Benchmark Code

Figure 6.9.: Desirable configurations of "phinally", "ivyep1", "hasep1" and "emmy" platforms for minimum energy consumption and minimum energy delay product (EDP) with and without compromising performance.

slowdown is independent of linearity of the chip power in the frequency. Moreover, Haswell does not have a memory bandwidth degradation with lower clock speed (as the others all do). This is why both the minimum energy and the minimum EDP cases have the same frequency of 1.2 GHz for jacobi benchmark.

The comparison of all results in terms of energy delay product shows that the first three architectures ("phinally", "ivyep1" and "emmy") are comparable but "hasep1" has a really very small energy delay product (see Fig. 6.9). This smaller EDP is justified by its highest performance due to the FMA units (doubled peak performance) in scalable region.

Chapter 7

Statistical Variation of Power Characteristics

An analysis has been done to obtain a statistical variation of the power characteristics for Ivy Bridge EP chip in "emmy" production system. For this purpose, the power profiles were analysed by using Intel RAPL counters from a large pool of characterization experiments, in which a machine operates at a varying CPU cores, n and clock frequency, f. This enables us to define a policy for power aware scheduling on "emmy" cluster and thereby saves energy cost of "emmy" cluster. Detail characteristics of this "emmy" system are available in Section 3.4.

7.1. Methods and results

7.1.1. Measurement methodology

The jobs with a shell script (see Sect. 3.5.2) were submitted on every emmy cluster node through the batch system:

```
1 #!/bin/bash
2
3 # List all available emmy nodes
4 NODES=`pbsnodes -a | grep '^e' | tail -n +17`
5
6 for n in $NODES; do
7 qsub -l nodes=${n}:ppn=xx,walltime=HH:MM:SS script.sh
8 done
```

The list of cluster nodes can be obtained by "pbsnodes -a" (line 4). It provided the 1120 data sets (one for each socket) of the power and the energy consumption using likwid-perfctr "ENERGY" group. Moreover, the power model parameters W_i were obtained by Levenberg-Marquardt non-linear least square curve fitting method (discussed in Section 3.5.3). The graphical representation of the distribution of data was done using a histogram.

Histogram

A statistical method is needed to elaborates the variation of the power characteristics. Histogram [28] is a graphically summarize representation of a frequency distribution of continuous data, where the equal-sized bins represent ranges of data, and the areas of the rectangles are proportional to the corresponding frequencies or the number of hits for each bin. It is used to better understand how frequently or infrequently certain values occur in the measured power

and its model parameter data. Thus, a histogram h_i is a function that counts the number of bins b for the total number of observations n to encounter the following conditions:

$$n = \sum_{i=1}^{b} h_i \tag{7.1}$$

Probability density function (pdf)

Histograms [28] give a rough sense of the density of the data, and estimation of the probability density function of the underlying variable is required. The density estimate is drawn as a curve rather than a set of bars. The normalized number of hits is the ratio of the bin count to the number of observations times the bin width.

$$\textit{Normalized number of hits} = \frac{b}{n * w} \tag{7.2}$$

In this section, a relative histogram was built for performing a statistical analysis of power consumption on "emmy". It normalized the number of hits and made the total area under histogram equal to one by modelling probability density function. A smoother probability density function was obtained, which reflected more precisely the distribution of the underlying power dissipation.

Bin selection (count and width)

There is no best way for finding number of bins and bins width, since it strongly depends upon the actual data distribution and the goals of the analysis. However, some scholars have tried to define a finest number of bins, but these approaches normally make strong assumptions about the shape of the distribution. Thus, the bin count can either be defined arbitrarily or via some useful methodical rule. The bin count b can be calculated from a proposed bin width w via ceiling function as:

$$b = \frac{[\max(x) - \min(x)]}{w} \tag{7.3}$$

Square-root choice $b = \sqrt{n}$ takes square root of the number of data points in the sample.

Scott's normal reference rule [28] $w = \frac{3.5\sigma}{\sqrt[3]{n}}$, where σ is the sample standard deviation, is optimal as it reduces the integrated mean squared error of the density estimate for random samples of normally distributed data. The Scott's result is true for large sample size n.

Algorithm to create a Histogram with pdf

The following procedure was done to overlay the probability density function on the top of the histogram:

1. Firstly, the minimum, maximum, average and standard deviation of data points related to the power and its model parameters W_i were determined.
2. The number of bins were established via Scott's normal reference rule for data points related to 544 compute nodes and via square root choice for 16 accelerator nodes data.
3. Bin width w was calculated via ceiling function as $b = \frac{[\max(x) - \min(x)]}{w}$

4. The absolute frequency was found by using the function "frequency" for all bins.
5. The relative frequency was determined by dividing the absolute frequency of one bin by the number of all observations times bin width w.
6. A histogram graph of the relative frequency was created and adjusted.
7. To overlay probability density function on top of histogram, the normal distributions were calculated for points starting at -3σ and increased it all the way up to +3σ.
8. The mean, bin width w as well as +/- 3σ, +/- 2σ and +/- 1σ values were displayed on histogram graph.

MATLAB Code

This matlab function was used to create a relative histogram using different input data and it delivers the probability density function of the fitted distribution as an output (see Appendix B.2).

The function "ecdf" in line 7 computes the empirical cumulative distribution function and "ecdfhist" (line 10) returns the bar heights and the position of the bin centers for equally spaced bins (determined via scott's rule). It normalizes the bar heights so that the area of the histogram is equal to 1. The "bar" function (line 11) creates a bar graph named as "hist". Further, lines 18-20 create a grid where function need to be computed while generating linearly spaced 100 points by "linspace" function. Moreover, the function "pdf" returns the probability density function of the probability distribution object created by fitting the normal distribution to the data (line 23 and 24).

7.1.2. Statistical Results

A study of the variation in the power characteristics for both CPU and DRAM modules is done with using realistic parallel applications on both compute and accelerator nodes of the "emmy" platform.

1. Emmy's compute Nodes

i. CPU power and model Parameters

A statistical analysis on the package power consumption of whole Ivy Bridge EP processor for 544 emmy's compute nodes was obtained in succession with above mentioned methodology to see how close they come to each other. The histogram plots for the CPU power dissipation and its model parameters W_i, $i = 0$, 1 2, are plotted on each 10 cores socket in turn with base clock speed f_0 of 2.2 GHz for both DGEMM and Jacobi benchmark applications, respectively. The histogram charts include outliers and show the specified bin widths calculated via Scott's rule, the mean data value and both positive and negative standard deviations (see Figures 7.1, 7.2).

These histograms characterize the CPU power dissipation data with a well-defined peak that is close in value to the mean. In addition, there are "outliers" of relatively very low frequency or density which were eliminated while performing analysis of power variation characteristics on "emmy" platform.

The tabulated information obtained from these histograms (see Table 7.1) excludes "outliers" and shows that the results inside the node for both socket 0 and 1 are very similar (i.e., maximum 0.02 standard deviation difference); whereas a large data variation was observed across nodes on

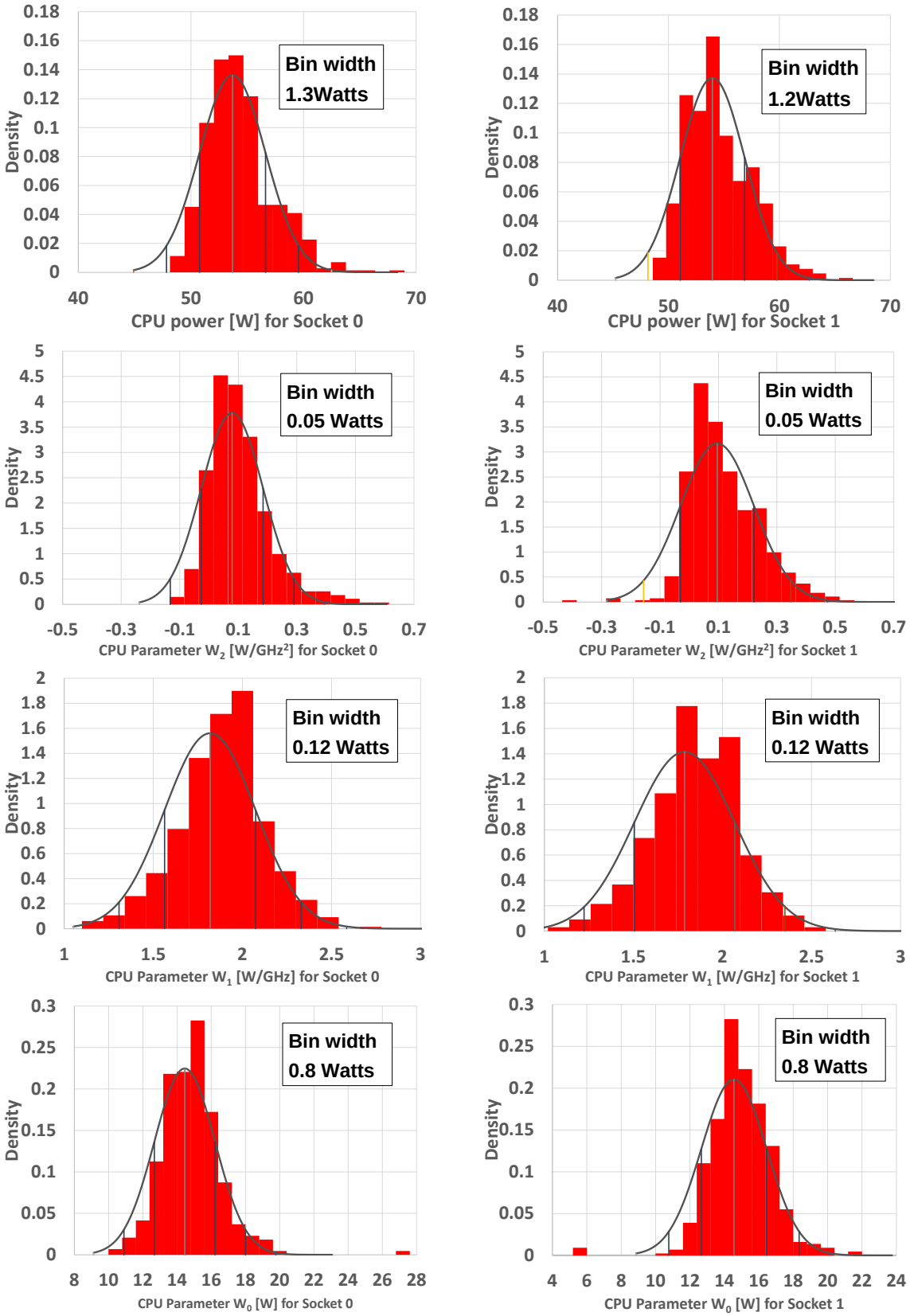


Figure 7.1.: The CPU power dissipation and its model parameters W_i for DGEMM benchmark on "emmy" system

same "emmy" platform. For both scalable and saturating application, the smaller CPU power dissipation W^{CPU} is according to expectations along with some measurement error because of

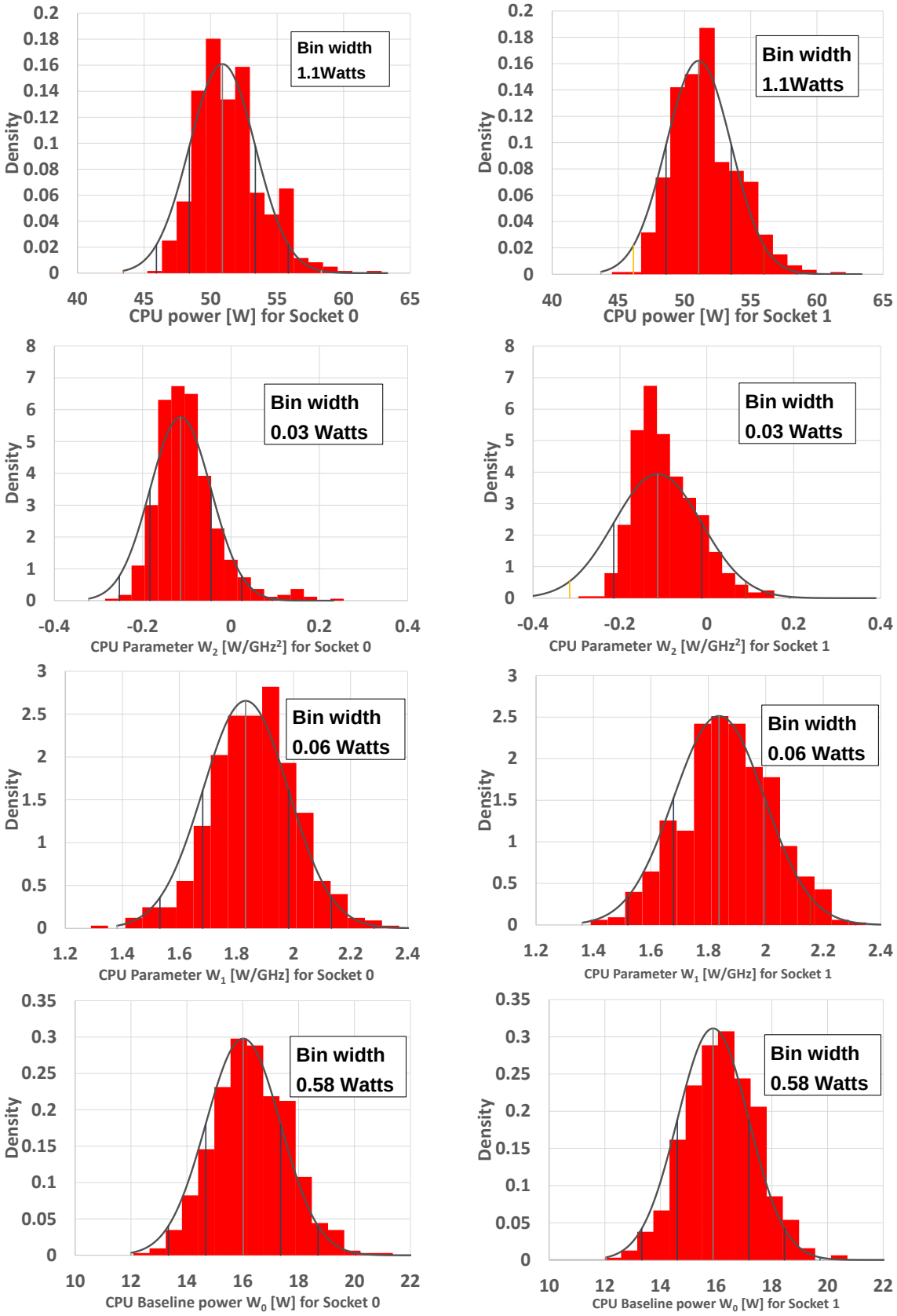


Figure 7.2.: The CPU power dissipation and its model parameters W_i for Jacobi benchmark on "emmy" system

lower base frequency of the "emmy" system. In addition, the baseline power $W_0 = W_{00} + 10W_{01}$ is constant on fixed 10 cores.

Table 7.1.: A statistics of CPU Power and its model Parameters W_i on "emmy" compute nodes at 2.2 GHz base frequency

	Mean		Standard deviation		Range	
	S0	**S1**	**S0**	**S1**	**S0**	**S1**
Jacobi CPU power $\mathbf{W}^{CPU}$ [W]	50.88	51.07	2.48	2.46	-1.9 – 2.1 SD	-1.8 – 2.2 SD
Jacobi CPU power parameter $\mathbf{W}_2$ [$\frac{W}{GHz^2}$]	-0.11	-0.11	0.07	0.07	-1.6 – 2.3 SD	-1.6 – 2.3 SD
Jacobi CPU power parameter $\mathbf{W}_1$ [$\frac{W}{GHz}$]	1.83	1.84	0.15	0.16	-2.3 – 2.3 SD	-2.0 – 2.5 SD
Jacobi CPU power parameter $\mathbf{W}_0$ [W]	16.02	15.87	1.34	1.35	-2.0 – 2.8 SD	-2.0 – 2.4 SD
DGEMM CPU power $\mathbf{W}^{CPU}$ [W]	53.69	53.69	2.93	2.91	-1.8 – 2.5 SD	-1.8 – 2.3 SD
DGEMM CPU power parameter $\mathbf{W}_2$ [$\frac{W}{GHz^2}$]	0.08	0.1	0.11	0.13	-1.5 – 2.2 SD	-1.4 – 2.1 SD
DGEMM CPU power parameter $\mathbf{W}_1$ [$\frac{W}{GHz}$]	1.82	1.79	0.26	0.28	-1.9 – 2.3 SD	-1.9 – 2.0 SD
DGEMM CPU power parameter $\mathbf{W}_0$ [W]	14.45	14.56	1.78	1.9	-2.0 – 2.9 SD	-1.5 – 1.8 SD

It was already discussed in Section 6.4.2 that "emmy" production cluster shows an exceptional linear power frequency trend and implies a zero quadratic factor W_2 in multi-core CPU power model because of its lower base clock speed ($f_0 = 2.2$ GHz). In addition, it has been observed from a pool of histogram plots that the quadratic chip power model parameter W_2 has comparatively a bit higher value for relatively hotter nodes with the large overall CPU power dissipation. However, for other nodes, which are comparatively a bit cooler, quadratic factor W_2 has values very close to zero which shows a perfect linear chip power and frequency behaviour. For example, node 27 on rack 9 (e0927) is one of the hottest node of "emmy" cluster out of 544 compute nodes and has high value of the quadratic factor W_2 parameter. Thus, in emmy cluster, chip power model could exhibit at times a perfect linearity trend or sometimes a slight bit quadratic behaviour depending on the temperature of specific chip.

It should be noted that all measurements for the statistical analysis were done using the CPU's RAPL counters. Any systematic inaccuracy in those counters, as well as manufacturing tolerance, environmental conditions and any variation in the voltage regulator circuitry on the motherboards may lead to the observed effects. A thorough investigation of the true reasons is beyond the scope of this thesis; one may, e.g., move CPUs from a "hot" node to a "cold" node and observe whether the "hotness" moves with the chips or stays with the board.

ii. DRAM power and model Parameters

The histogram charts for the DRAM power dissipation and its model parameters (W_0^{DRAM}, w) are shown with indication of the mean values, standard deviations σ and bin widths w for both scalable DGEMM and saturating Jacobi application, respectively (see Figures 7.3, 7.4).

Table 7.2.: A statistics of DRAM Power and its model Parameters on "emmy" compute nodes at 2.2 GHz base frequency

	Mean		Standard deviation		Range	
	S0	**S1**	**S0**	**S1**	**S0**	**S1**
Jacobi DRAM power $\mathbf{W}^{DRAM}$ [W]	41.99	44.41	1.32	1.47	-2.1 – 2.2 SD	-1.2 – 2.0 SD
Jacobi DRAM power parameter w [$\frac{Ws}{GB}$]	0.64	0.67	0.03	0.03	-2.0 – 2.3 SD	-1.7 – 2.1 SD
Jacobi DRAM power parameter $\mathbf{w}_0^{DRAM}$ [W]	16.39	17.73	1.26	1.3	-2.0 – 2.0 SD	-1.9 – 2.0 SD
DGEMM DRAM power $\mathbf{W}^{DRAM}$ [W]	17.93	19.34	1.28	1.21	-1.8 – 2.1 SD	-2.0 – 2.0 SD
DGEMM DRAM power parameter w [$\frac{Ws}{GB}$]	1.11	1.16	0.04	0.04	-1.7 – 2.2 SD	-1.9 – 2.0 SD
DGEMM DRAM power parameter $\mathbf{w}_0^{DRAM}$ [W]	10.57	11.65	1.25	1.15	-2.0 – 2.1 SD	-2.0 – 2.1 SD

The information obtained from these histograms of DRAM power is tabulated in Table 7.2. Again the DRAM power results inside the node for both S0 and S1 are very similar (i.e.,

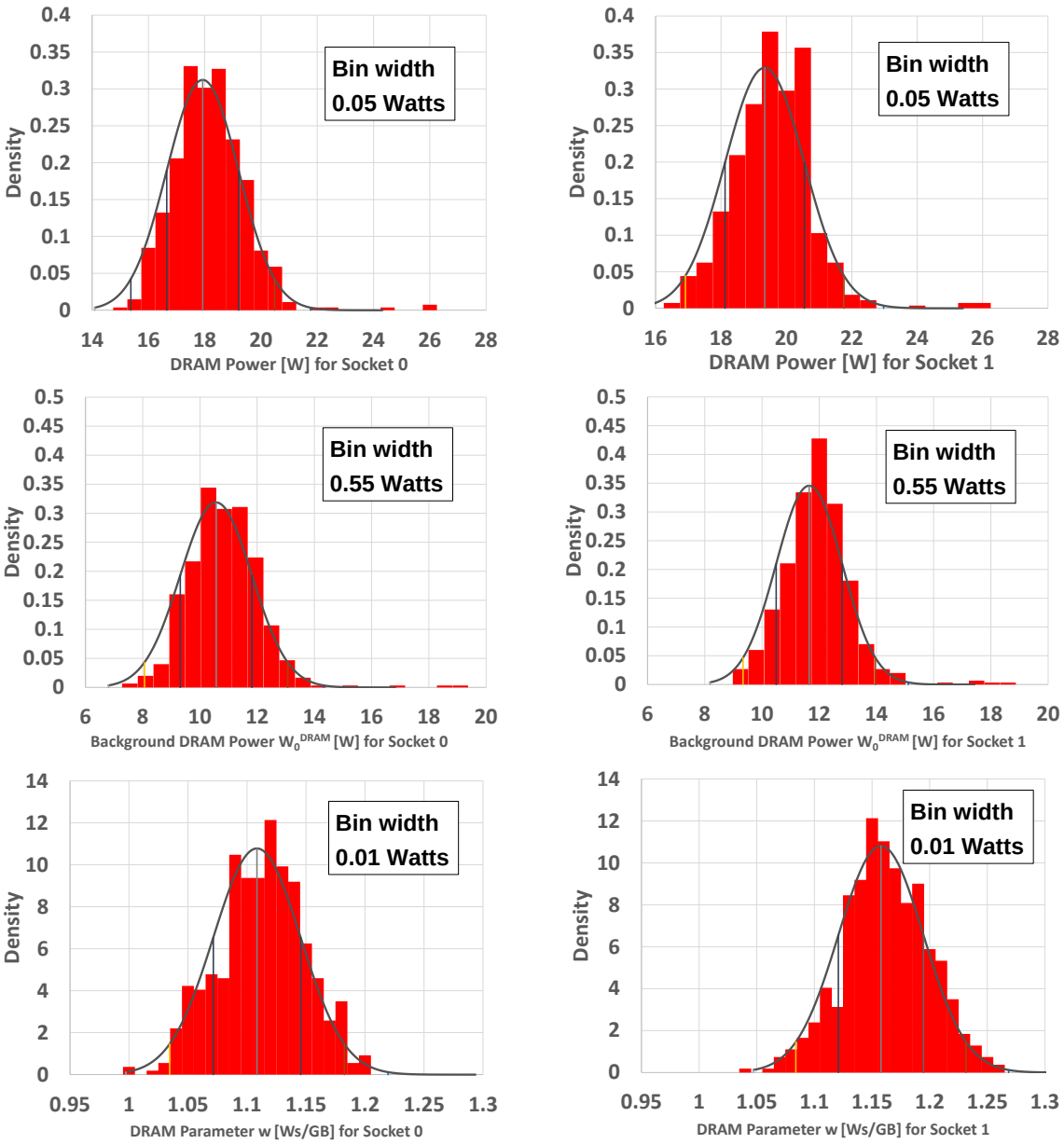


Figure 7.3.: The DRAM power and its model parameters (w and W_0^{DRAM}) for DGEMM benchmark on "emmy" platform

maximum 0.15 standard deviation difference) and a considerable data variation across the chip was observed on same "emmy" platform.

The observation of results exhibit that a node with high DRAM power dissipation W^{DRAM} will exhibit a higher background DRAM power W_0^{DRAM}. For example, among all 544 emmy compute nodes, node "e0116" dissipates highest DRAM power and has high background DRAM power dissipation W_0^{DRAM} for running parallel applications.

2. Emmy's Accelerator Nodes

The accelerator nodes on "emmy" are interesting to analyse, since these nodes have the same amount of DRAM but faster DRAM frequency. The accelerator nodes were allocated by specifying the property "ddr1866" upon job submission.

The histogram plots of both CPU and DRAM power dissipation (W^{CPU}, W^{DRAM}) and their

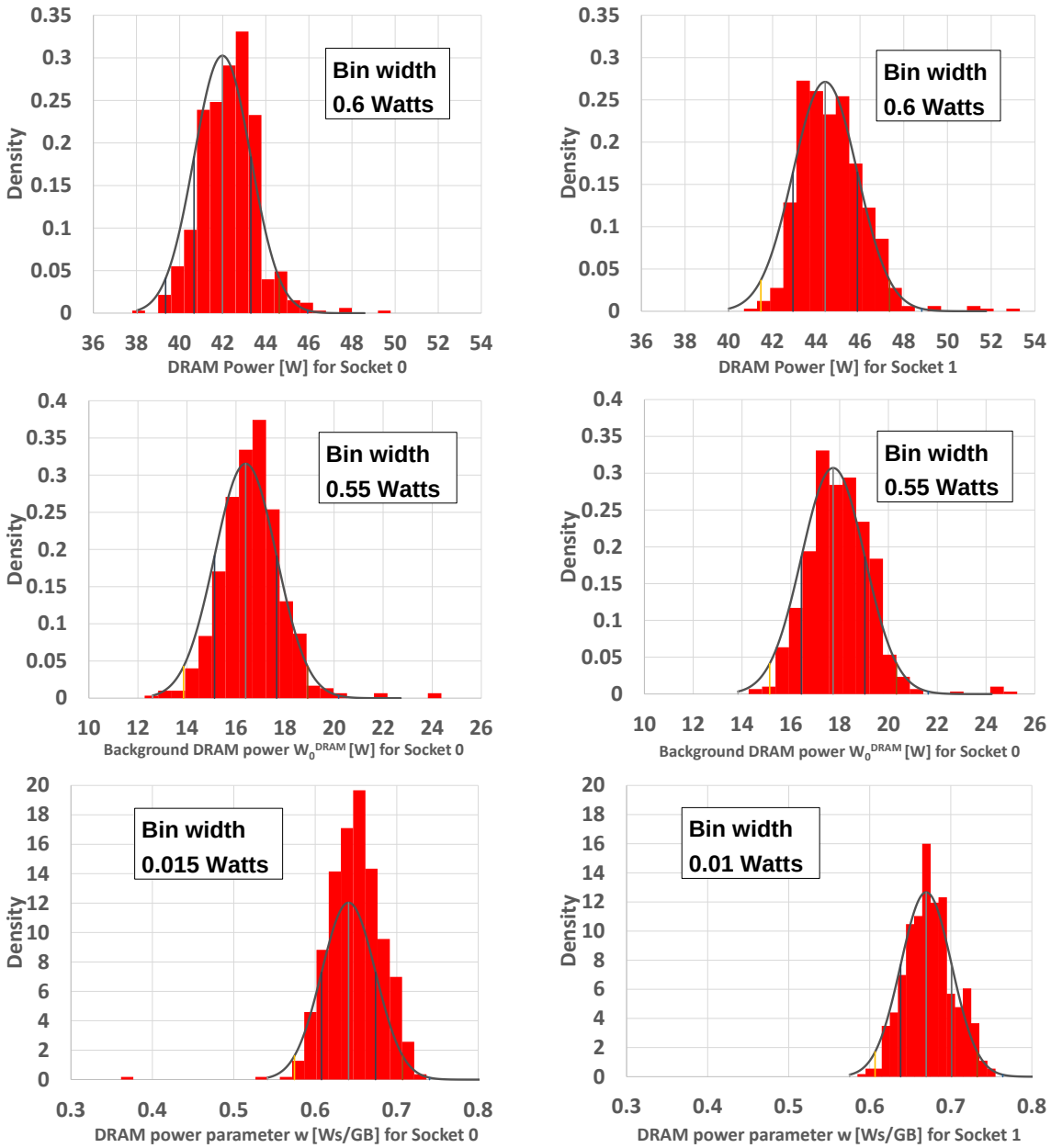


Figure 7.4.: The DRAM power and its model parameters (w and W_0^{DRAM}) for Jacobi benchmark on "emmy" platform

respective model parameters ($W_2, W_1, W_0, W_0^{DRAM}$ and w) are shown in Appendix D. As 16 accelerator nodes result in 16 data points for each sockets and the Scott's result for bin width selection is only true for large sample size so the Square-root choice formula was utilized for the bin width selection of the accelerator nodes.

The tabulated data of both CPU and DRAM modules shows that again the results inside the node are similar in contrast to the results across chip. Additionally, the CPU power dissipation data for both the compute and the accelerator nodes are comparable with slightly larger power for the saturating applications in the accelerator nodes.

However, a very small DRAM power dissipation was observed at accelerator nodes against compute nodes of "emmy". From previous experiments, it is known that the actual parameters (slope w, base power W_0^{DRAM}) of DRAM power model depend on the particular type and number of DRAM DIMMs, so we can only measure them (see Section 4.4). It is entirely possible that

we could buy an emmy node with the same performance characteristics but much less DRAM power consumption because somebody has chosen a different DIMM manufacturer (see Tables 7.3, 7.4).

Table 7.3.: A statistics of CPU Power and its model Parameters on "emmy" accelerator nodes at 2.2 GHz base frequency

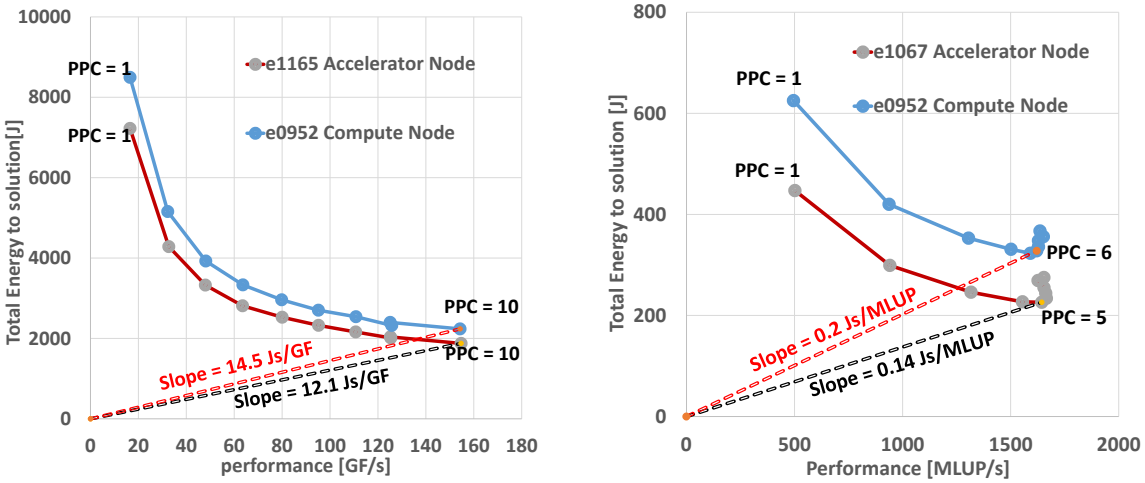
	Mean		Standard deviation		Range	
	S0	**S1**	**S0**	**S1**	**S0**	**S1**
Jacobi CPU power W^{CPU} [W]	54.98	55.21	4.91	5.34	-0.6 – 1.7 SD	-0.6 – 2.2 SD
Jacobi CPU power parameter w_2 $[\frac{W}{GHz^2}]$	-0.07	-0.06	0.1	0.08	-0.9 – 1.8 SD	-0.8 – 1.4 SD
Jacobi CPU power parameter w_1 $[\frac{W}{GHz}]$	1.98	1.95	0.2	0.18	-2.2 – 1.8 SD	-1.8 – 2.8 SD
Jacobi CPU power parameter W_0 [W]	14.95	15.06	2.63	2.17	-0.6 – 2.3 SD	-1.0 – 2.7 SD
DGEMM CPU power W^{CPU} [W]	53.54	53.49	12.17	13.11	-0.6 – 2.6 SD	-0.6 – 2.3 SD
DGEMM CPU power parameter w_2 $[\frac{W}{GHz^2}]$	0.13	0.14	0.12	0.13	-1.0 – 1.6 SD	-0.9 – 2.2 SD
DGEMM CPU power parameter w_1 $[\frac{W}{GHz}]$	1.43	1.44	0.59	0.59	-1.0 – 2.9 SD	-0.5 – 2.0 SD
DGEMM CPU power parameter W_0 [W]	15.49	15.29	3.44	3.1	-1.0 – 2.3 SD	-0.1 – 2.5 SD

Table 7.4.: A statistics of DRAM Power and its model Parameters on "emmy" accelerator nodes at 2.2 GHz base frequency

	Mean		Standard deviation		Range	
	S0	**S1**	**S0**	**S1**	**S0**	**S1**
Jacobi DRAM power W^{DRAM} [W]	19.64	19.84	4.17	4.66	-0.7 – 2.1 SD	-0.6 – 1.8 SD
Jacobi DRAM power parameter w $[\frac{Ws}{GB}]$	0.27	0.27	0.08	0.08	-0.4 – 0.2 SD	-0.5 – 0.1 SD
Jacobi DRAM power parameter w_0^{DRAM} [W]	8.59	8.62	1.07	1.37	-0.9 – 1.8 SD	-1.3 – 1.6 SD
DGEMM DRAM power W^{DRAM} [W]	8.5	8.54	2.09	2.41	-0.9 – 2.1 SD	-0.7 – 1.9 SD
DGEMM DRAM power parameter w $[\frac{Ws}{GB}]$	0.49	0.5	0.15	0.16	-0.6 – 2.9 SD	-0.6 – 3.0 SD
DGEMM DRAM power parameter w_0^{DRAM} [W]	5.39	5.38	0.82	0.97	-0.8 – 2.9 SD	-1.8 – 2.2 SD

7.2. Consequences for code execution

Even on the same "emmy" platform, a strong variation of the power dissipation parameters



(a) DGEMM Benchmark Code **(b)** Jacobi Benchmark Code

Figure 7.5.: A comparison of energy consumption in terms of "Z-plot" of a average power compute node and accelerator node at 2.2 GHz base frequency on "emmy" platform

across the chip was observed which has a significant impact on consequence for code execution.

Further, the "outliers" in statistical histogram method are the nodes with very high CPU or DRAM power dissipation and they lead to the higher values of their respective model parameters. Assuming the power readings are reliable, these "outliers" need to be avoided for running an application on a system in most power efficient way. For example, especially for scalable applications, 27th node of rack 9 (e0927) is one of the hottest processor of "emmy" production cluster and is never suggested for a power efficient simulation run.

Depending upon temperature of chip, we can arrange cool nodes, hot nodes and outliers in a sequence that every time cool node will be allotted as a first preference to achieve the goal of power capping on "emmy" cluster. This arrangement of nodes can lead to overall less power dissipation of "emmy" cluster, which ultimately can increase its operation runtime [29].

A comparison of energy consumption of an average temperature (neither hotter nor cooler) regular compute and accelerator node is shown in Fig. 7.5. It illustrates that the performance of both type of nodes is more or less identical. However, the accelerator nodes on "emmy" clusters are more desirable against regular compute nodes for energy-efficient execution of parallel applications (especially for saturating cases). This is because of much smaller DRAM power dissipation of the accelerator nodes.

Chapter 8

Connection to Microscopic Power Models

Micro benchmarks with tuneable intensity were created in assembly language using "likwid-bench" tool to achieve a very high performance or bandwidth as close to roofline model as possible. All micro benchmarks (see Appendix C.4) were made readable only (one load) to get the best memory bandwidth. A variation in the intensity was achieved by increasing flops to bytes ratio on "phinally" test system at 2.7 GHz base clock speed.

A lot of researchers have defined the concept of machine balance B_m [25], [30]–[34] as a ratio of the number of memory operations per CPU cycle to the number of floating-point operations per CPU cycle for a particular processor. Whereas, the "computational intensity" I_c applies to computational kernels, compares the floating-point work required with the number of memory references. The eight cores peak performance P_{peak} of Sandy Bridge EP processor in "phinally" system is 172.8 GFlops/s which leads to a machine intensity I_m of 3.9 F/B or 31.2 F/W. The tuneable intensity benchmark achieves 171.05 GFlops/s (98.98% of system peak double-precision performance) for highly intense code with no data transfer. Whereas, saturated memory bandwidth b_s of 43.6 GBytes/s was achieved for lower intensity memory bound benchmarks. Figure 8.1 shows that variable intensity benchmarks behave in accordance with the Roofline model.

Methodology

The LIKWID suite also contains a micro-benchmarking framework "likwid-bench" [9], [19] that runs assembly language benchmarks without the uncertainties of compiler code generation. In present work, the "likwid-bench" was built with "likwid-perfctr" counter support so it measures only regions and not end-to-end measurements.

```
1 likwid-perfctr -m -g ENERGY -c N:0-7 ./likwid-bench -t my_bench -i 100 -g 1 -w ↩
    S0:1GB:8:1:2
```

This example runs the benchmark "my_bench" for a working set size of 1 GB on all eight physical cores of socket 0 on "phinally" system by specifying chunk size 1 and stride 2. It is highly recommended to have a significant run-time for the actual kernel inside a program. It is more likely that there is overhead involved in starting/stopping the counters. This overhead [35] can be quite considerable, so the benchmark should run for a long time.

The basics of x86-64 ISA includes 16 floating point single instruction multiple data (SIMD) registers, that is, "xmm0-xmm15" for 128 bit SSE and "ymm0-ymm15" 256 bit AVX. The description of included header for assembly code in "likwid-bench" is shown in Table 8.1. For the

Table 8.1.: Header of assembly code in "likwid-bench". All parameters are the same for all intensity benchmarks, except FLOPS.

Header	Description
STREAM 1	One load stream
TYPE DOUBLE	How many bytes are transferring
FLOPS 2	Number of flops per iteration
BYTES 8	Number of bytes transfer per iteration (1 LD)
LOOP 16	Number of iterations inside loop iterations

AVX vectorized assembly benchmark, Sixteen scalar iterations in the loop body are reflected by the four wide execution and the four way unrolling. Whereas, the "LOOP 8" for the SSE instruction is justified by the two floating point instructions per cycle and the unroll by four. Moreover, the assembly codes in "likwid-bench" always have memory addresses in square brackets with GPR *8, which is a double precision integer register.

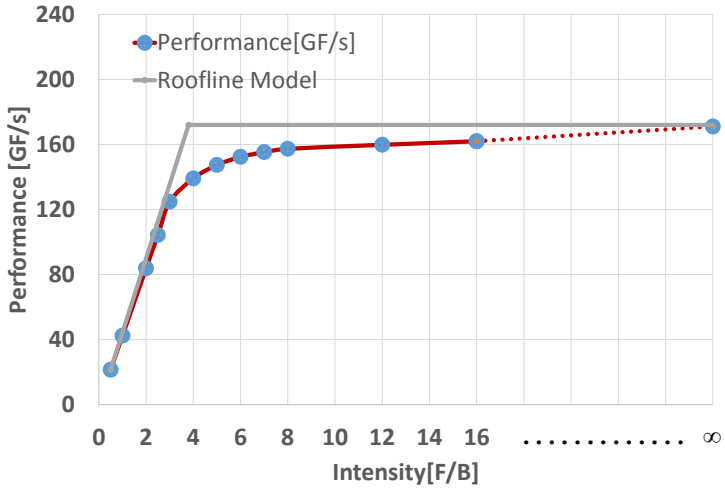


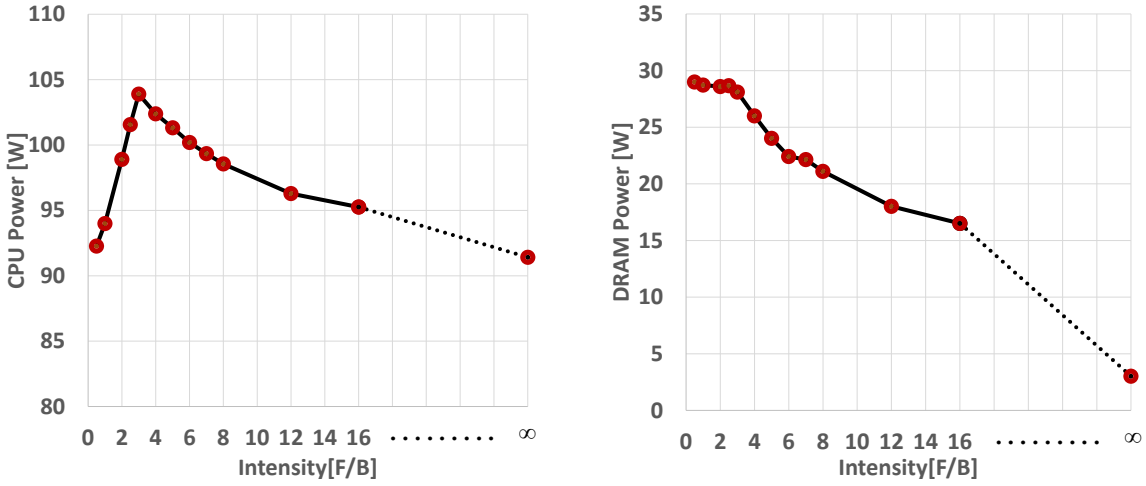
Figure 8.1.: Performance of variable intensity benchmarks with the theoretical Roofline as the upper limit at 8 cores and 2.2 GHz of the "phinally" system

8.1. Tunable Intensity effect on CPU and DRAM Power model

8.1.1. CPU and DRAM Power versus intensity characterisation

The power consumption of recent processors depends strongly on the currently running task. There are "hot" tasks, which lead to high power dissipation, and "cool" tasks, which cause lower power dissipation. Hence, if the processors of a multiprocessor system are executing different tasks, they are likely to have different temperatures.

For lower intensity/memory-bound case, lower CPU power is reflected the fact that the processor mostly remains idle during code execution to await data while a lot of power is burnt in the memory hierarchy resulting in large DRAM power. With an increment in intensity, the increased work per second results in growth of CPU power while the power in memory hierarchy do not vary because of saturated memory bandwidth. When reaching near knee of roofline curve, both CPU and DRAM modules are fully utilized with the peak performance per core and

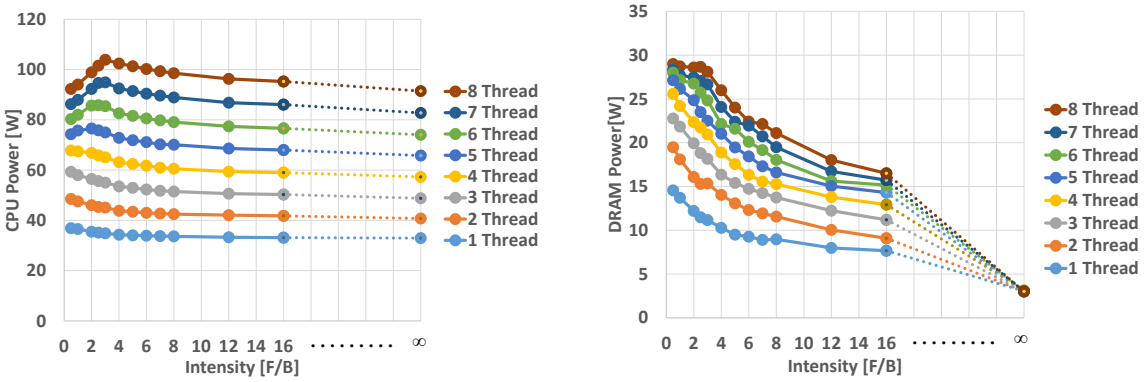


(a) CPU Power-Intesity curve **(b)** DRAM Power-Intesity curve

Figure 8.2.: The CPU and DRAM power versus intensity at 8 cores of the "phinally" system

saturated memory bandwidth. A CPU cannot become more hotter than this value. Afterwards, the power in memory subsystem decreases as benchmarks decouple from the memory bound to the compute-bound case with less data transfers.

There always exists some overlap between the core power and the data transfer power and they are never completely separate. For example, zero intensity copy benchmark should only include data transfer and it does not do any actual work. However, execution units are sort of busy as they still need LD and ST units, so beyond data transfer some power is spent on LD/ST and integer (used to count loop counters) pipelines. Thus, it can be observed that for infinite intensity/no data-transfer case, most power is burnt in the computation units whereas for zero intensity/no computation case, the most power is consumed by data transfer.



(a) CPU Power-Intesity curve **(b)** DRAM Power-Intesity curve

Figure 8.3.: The CPU and DRAM power versus intensity with varying number of active cores on the "phinally" system

In Fig. 8.3(a), it is interesting to note that the peak of the curve, which is close to knee of roofline curve, shifted towards left with a decrease in the number of threads. This peak vanishes when the number of threads is below 5, as the performance decreases and the memory bandwidth cannot be saturated any more (in other words, the previously saturating characteristic of the benchmark changes to a scalable characteristic).

DRAM power (see Fig. 8.3(b)) depends upon the bandwidth drawn from dynamic random access memory (DRAM). Thus, for higher thread count (7 or 8), DRAM power remained same as long as the bandwidth was saturated. When benchmarks decouple to the compute bound case at about 5 threads, DRAM power falls off quickly with rising intensity. It is noteworthy that there is a region of thread count (5 or 6) and intensity where the memory bandwidth is still saturated but the DRAM power already falls off. This is due to the different memory access characteristics when running with different numbers of threads: More threads mean more concurrent data streams from the memory DIMMS, i.e., more active transistors in the DRAM. This is another positive effect (in terms of energy consumption) of not using all cores if the loop performance saturates across cores.

Figure 8.4 shows ultimately the total power which is a contribution of both CPU and dynamic random access memory.

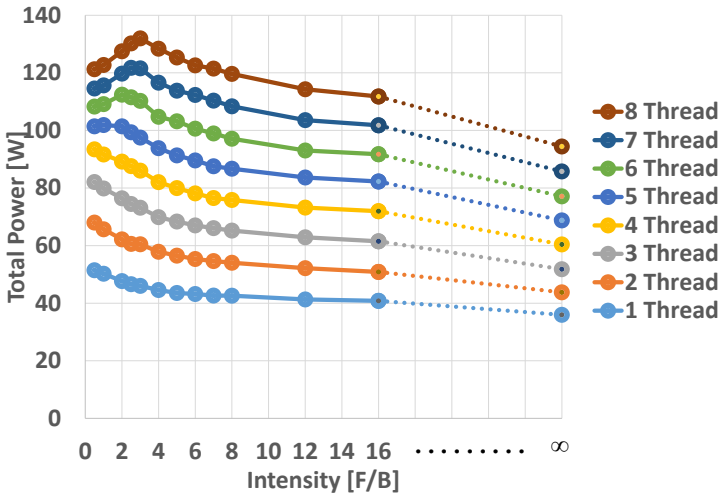


Figure 8.4.: The total power dissipation versus intensity with varying number of active cores on the "phinally" system

8.1.2. CPU Power model parameters versus intensity characterisation

1. Quadratic factor W_2 – intensity relation

By decoupling the benchmark from the memory bound code to the compute bound code, the performance does not change any more so that quadratic factor of power W_2 remains constant independent of the code intensity (see Fig. 8.5). It has a kind of asymptotic behaviour by extrapolation to highest intensity benchmark.

2. Linear factor W_1 – intensity relation

Figure 8.6(a) shows that once a code decouples from the memory bound code to the compute bound code, the power intensity behaviour will start to become more quadratic and less linear. moreover, from power model $W^{CPU} = W_0(n) + (W_1 f + W_2 f^2)n$, it is obvious that the quadratic term dominates always (even for low intensity) due to the f^2 factor, and dominates even more so when the intensity is high.

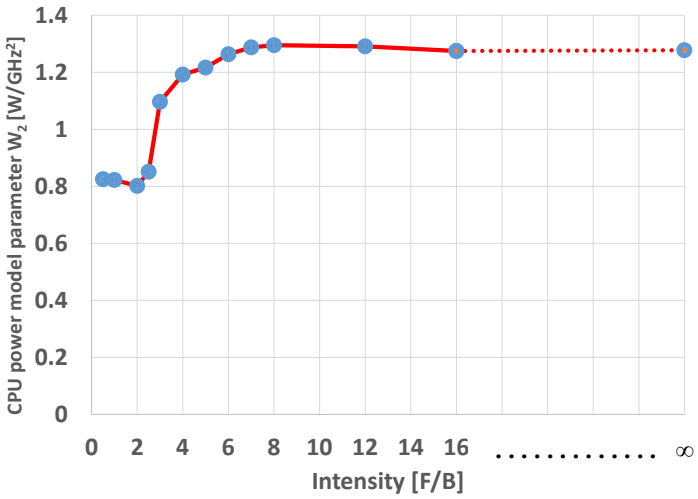


Figure 8.5.: CPU power model parameter W_2 versus intensity at 8 cores of the "phinally" system

3. Baseline power W_0 – intensity relation

The baseline power W_0 is a contribution that comes from substrate or from a part not participating into computation and consists mainly of the power dissipation caused by leakage currents. The power model (5.1) assumes that the baseline power W_0 varies with switching on/off the different number of cores irrespective of the type of running code. For example, for phinally architecture ($W_{00} = 14$ W, $W_{01} = 1.2$ W ; see Sect. 5.1), the baseline power W_0 is 15.6 W for one core and 24.6 W for eight cores. Figure 8.6(b) shows that the results for the baseline power W_0 at single and eight cores are as predicted by the power model (5.1). However, the unexpectedly low baseline power W_0 at left side of Roofline knee for eight cores might be justified by the low processor temperature at lower intensity.

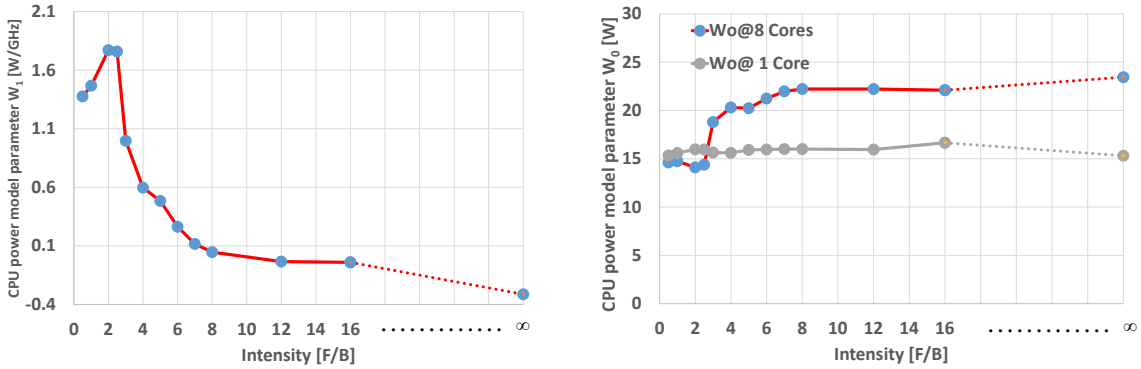


Figure 8.6.: The CPU power model parameters W_0 and W_1 versus intensity at 8 cores of the "phinally" system

8.1.3. DRAM Power model parameters versus intensity characterisation

By decoupling the benchmark from the memory bound to the core bound case, performance remains maximum but data transfers start to decrease as shown in Figure 8.7. The decreasing data transfers causes a continuous decrease in power dissipated by the dynamic random access memory module which is reflected by rising w parameter and reducing background power W_0^{DRAM} (see Fig. 8.8). This background power reduction reflects an impression that it goes to some power saving state with decreasing memory bandwidth. We have already seen in Fig.

8.2(b) that the DRAM seems to dissipate more power if many different streams are used concurrently by multiple cores, even if the raw bandwidth is the same. It means our bandwidth-based DRAM power model (5.4) is somehow simple since it does not take into account how many different addresses are utilized in the RAM.

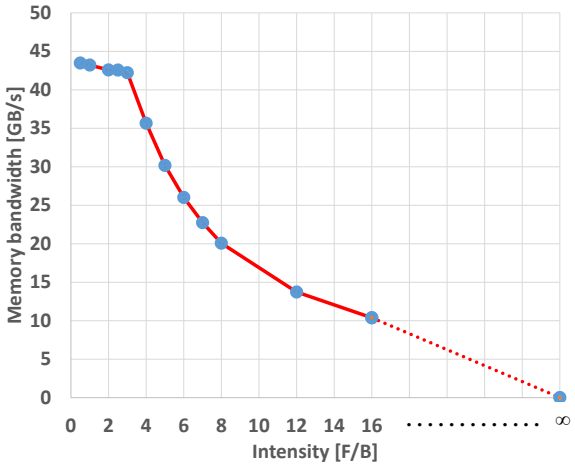


Figure 8.7.: Memory bandwidth versus intensity at 8 cores of the "phinally" system

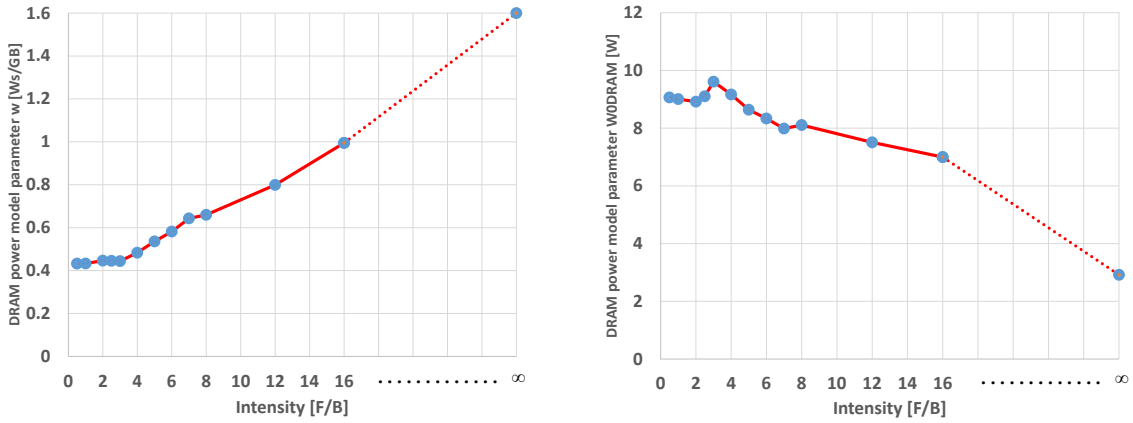
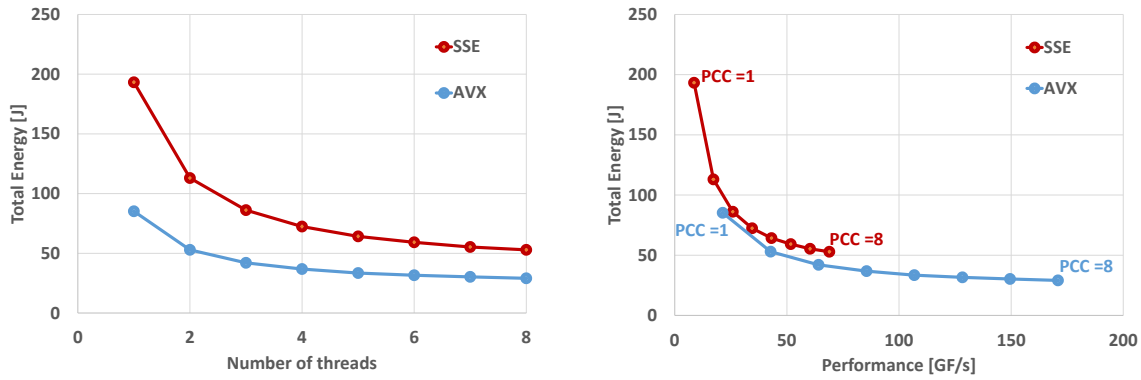


Figure 8.8.: The DRAM power model parameter (w and W_0^{DRAM}) versus intensity at 8 cores of the "phinally" system

8.2. Energy efficiency of different kind of instructions

The power consumption of recent processors depends strongly on the kind of instructions the processor executes. It is interesting thing to look at energy efficiency of different type of floating point instructions (e.g., AVX and SSE instructions; see Appendix C.4). An energy-efficiency comparison of AVX and SSE instructions was performed for highest intensity benchmark (4 Flops, 0 STREAM) with a working set size of 1 GB on "phinally" system. It is observed that the different type of instructions executed by processor strongly affect the energy consumption of the processors. For instance, running an application with SSE instructions gives much less performance compared to AVX (see Fig. 8.9(b)) with some power difference. This means that the SSE instruction does same work with taking larger time and energy than AVX instruction, and thus a lot of energy is saved by using AVX instructions. Hence, in terms of the energy consumption, the AVX flops are less expensive compare to the SSE flops.



(a) Energy vs. core count

(b) Energy vs. performace (Z-plot)

Figure 8.9.: An energy-efficiency comparison of AVX and SSE instructions for highest intensity benchmark (4 Flops, 0 STREAM) with a working set size of 1 GB at base frequency of 2.2 GHz on "phinally" system

8.3. Microscopic performance, power and/or energy models

Previously discussed models only consider global parameters and ignore the microscopic information. For instance, the previous power/energy measurements are the values that the system consumes for executing code; microscopically, code execution can be split into "real work", i.e., instructions, and data transfers. The fit parameters W_i cannot differentiate between those contributions, other than having different values for different computational intensities. There are several energy contributions from data transfers. These contributions are from memory to L3 cache, L3 cache to L2 cache and L2 cache to L1 cache as shown in Figure 8.10.

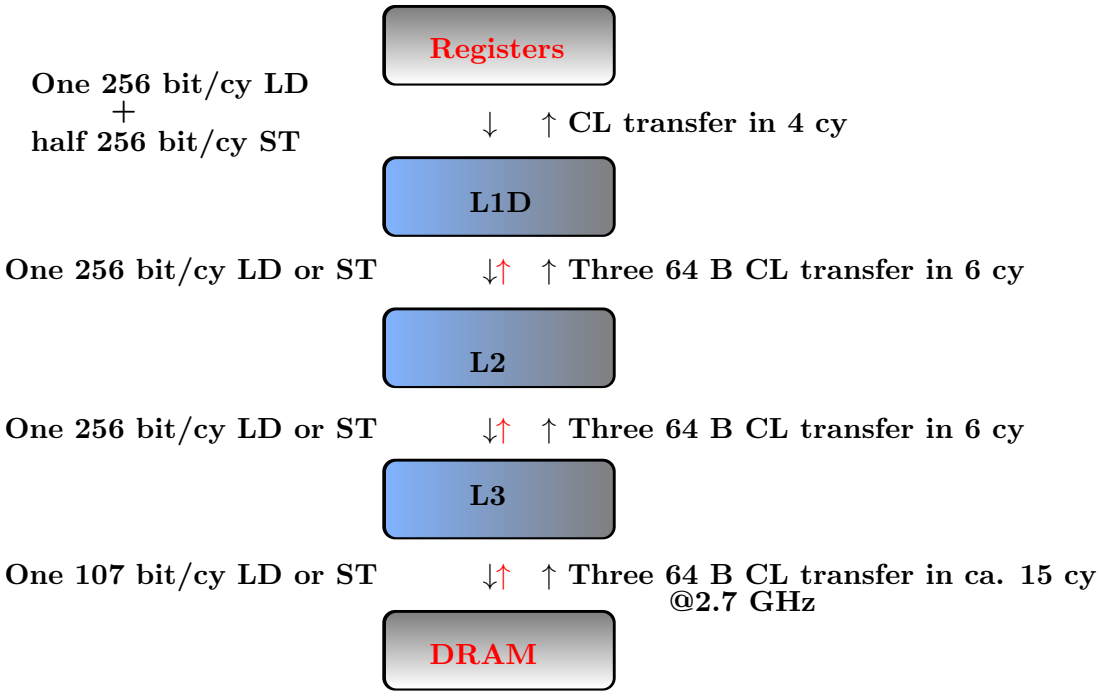


Figure 8.10.: Memory hierarchy with data transfer for copy benchmark on single core of the "phinally" system

The performance modelling on microscopic level was performed using "Execution Cache Mem-

ory (ECM)" model [10], [36]. The ECM model refines the roofline model [11] and develops a deeper understanding of the interaction of a benchmark with the hardware by adding more accurate description about memory hierarchy.

$$\textit{Simplified ECM model performance:} \quad P_{ECM}(n,f) = \quad min((1+\Delta\nu)nP_0, P_{roof}), \tag{8.1}$$

with $P_{roof} = min(P_{max}, \frac{b_s}{B_c})$. The term $(1+\Delta\nu)$ assumes that the single-core performance is linear in the clock speed, which is not true if there is a non-negligible runtime contribution from memory data transfers.

There are two main contribution for the dynamic power/energy modelling on the microscopic level: First is the number of watts/joules per byte π_{byte} or ε_{byte} which are measured in pW/B or pJ/B, and second is the number of watts/joules per flop π_{flop} or ε_{flop}, in pW/F or pJ/F.

Three benchmarks have been run to get the energy cost for a byte transfer (load/store) ε_{byte} or for computing a certain flop (addition/multiplication) ε_{flop} on different working sets from the memory up to the registers. These benchmarks have only flops and only data transfers, respectively. In reality, the energy measurements are in pico joules per CL, since data transfers between caches are in packets of one cache line and a complete cache line is always used to make full use of it. One may certainly break it down to pico joules per byte.

It was observed that when the data set comes from the L3 cache (10 MB), the DRAM power stays constant. While the L3 power increases with adding more cores, since the L3 cache is segmented per core and each core can use all segments that means the bandwidth increases with increasing cores count. Furthermore, when the data fits into L2 cache (100 KB) then there is no data transfer for L3 cache and memory. Thus, a well-controlled behaviour of data transfer through the memory hierarchy was utilized to find the energy cost per flop (ε_{flop}) and cost per byte transfer from MEM to L3 ($\varepsilon_{byte}^{DRAM-L3}$), L3 to L2 ($\varepsilon_{byte}^{L3-L2}$) and L2 to L1 ($\varepsilon_{byte}^{L2-L1}$). Moreover, when adding 4 flops with same data transfer as in copy benchmark, there is enough space in the computational pipeline and they execute in parallel and so runtime stays constant no matter weather the flops are done or not. This makes it possible to measure π_{flop} and ϵ_{flop} directly.

Limitation and correction

This suggested model gives a very rough estimation with the energy consumption when no activity is going on inside core. When the processor is idle and waiting for a data transfer, the amount of joules burnt because of leakage current depend on the waiting time. Thus, the energy consumption can split into two parts: First is the energy burnt for actual data transfers and for computations inside core and second is the consumed energy while waiting for data transfers in register or L1 cache.

It is a common thinking that the energy consumption of a system comes mainly because of the data transfers. It is true if we transfer a data for a short distance so it results in transistors switching and consumes energy. However, this impression seems not true when we do arithmetic on a core and need data from other node. It takes such a long time so most of the energy consumption is done while waiting data; so, the baseline energy contribution becomes significant along with data transfer on single core. However, in the end it looks as the data transfers are costly; whereas, in reality increased energy cost is reflected by both long waiting time for data while performing no computation and data transfers. Note that this is the situation with today's architectures; it is expected that future micro-architectures will burn much more power via the actual data transfer, which will be a major concern - but not today.

The concern is to measure the number of joules for a particular byte transfer and during this transfer time the whole chip consumes baseline energy E_0 which is burnt anyway even when processor is idle and waits for the data transfer. The baseline energy E_0 will be different for benchmarks having different runtime. Thus, a correction is applied by taking out this baseline energy and only considering the dynamic part of the energy consumption.

Table 8.2.: Single core of dynamic energy cost for one flop (tread addition and multiplication on the same footing) and for one byte transfer(load/store) with baseline power $W_0 = 15.9$ W on "phinally" system

Metrics	Energy cost				
	ε_{flop}	$\varepsilon_{byte}^{L1-REG}$	$\varepsilon_{byte}^{L2-L1}$	$\varepsilon_{byte}^{L3-L2}$	$\varepsilon_{byte}^{MEM-L3}$
Flop only	830 pJ/F	0	0	0	0
Load only	0	227 pJ/B	314 pJ/B	256 pJ/B	1880 pJ/B
Store only	0	377 pJ/B	300 pJ/B	340 pJ/B	2977 pJ/B

Validation

The energy consumption can be predicted for a variety of tasks with the multiple operations while knowing the microscopic level parameters (i.e., the energy cost of a single flop and a byte transfer). For validation of this hypothesis, a real benchmark like 2D jacobi was chosen which have a lot of data transfer with few flops so the dominant energy contribution is the data movement. A comparison of measurements with the analytically predicted energy consumption was done by putting together the knowledge about the data transfer cost, the flop cost, the ECM model and the layer condition in energy model. For Jacobi stencil code, three streams are needed to transfer when the layer condition is satisfied. Whereas, when layer condition is violated, so there are more data transfers (five streams) across expensive data paths and the code takes longer time which results in large energy cost.

Table 8.3.: Single core energy cost for one flop and energy cost for one byte transfer with baseline power $W_0 = 15.9$ W on "phinally" system

Metrics	Copy (AVX) (9 it * 1 GB)	Jacobi (lc_{L3}) (100 it *8k *8k)	Jacobi (lc_{L2}) (100 it *8k *8k with blocking)
Cycles per cache line	30.73	42.8	31.13
Memory Data volume (Byte)	9.66E+9	1.54E+11	1.54E+11
E_0 [pJ/B]	1414	1308	949
Calculated Total energy [J]	46	681	649
Measured Total energy [J]	45	651	638

The table 8.3 shows that how far the predicted values coincide with the measurements for both same and different number of cache line transfers through memory hierarchy. We observed that the single-core energy consumption of streaming kernels has a large baseline energy contribution into total energy compare to multiple cores. However, for multi-cores, the total energy becomes increasingly much more dominated by the dynamic power (which our power model also predicts).

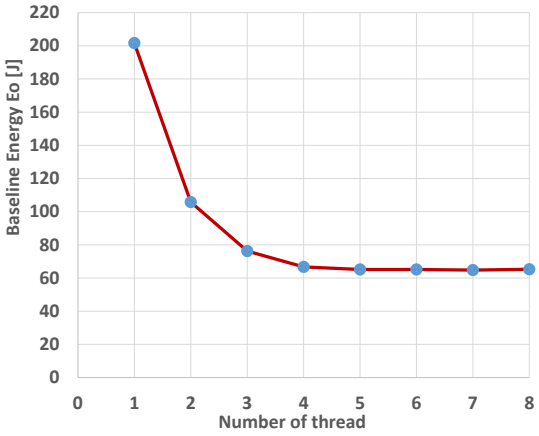


Figure 8.11.: The baseline energy E_0 at varying number of cores for jacobi stencil on "phinally" system

8.4. Vuduc parameters to W_i parameters

This section describes the connection between the current model and the arch line model described by Choi et al. [14]. Their approach towards cost calculation for short-running code paths is to fix low level microscopic parameters, such as the cost per flop and the cost per byte transfer (see Sect. 8.3). Their model computes the energy consumption of a benchmark with known amount of data and the cost for a data transfer. Whereas, the focus of present model (see Chapter 5) lies on predicting energy consumption of an algorithm which has large number of flops and data transfer by fixing different W_i parameters. This estimates the power and ultimately the energy consumption of an algorithm.

The goal of the present work is to marry these two approaches by deriving W_i parameters from low level microscopic parameters with different cache line transfers for the cache and the memory hierarchy. The energy cost for one flop and one byte data transfer and W_i parameters with varying intensities are already known from previous sections 8.1 and 8.3. The baseline power W_0 is constant for single core irrespective of type of running task (see Fig. 8.6(b)). The quadratic factor W_2 is a fusion of many effects (i.e., work inside core and the data transfer) and it varies with number of cores. For full eight cores of "phinally" system, W_2 is different for both memory-bound case (doing less work inside core and more data transfers) and compute-bound case (doing more work inside core and less data transfers) as shown in Fig. 8.5. However, when we visualize a single core, W_2 remains more or less same for both memory-bound and compute-bound case because of vanished Roofline peak at single core. The linear parameter W_1 is the factor that decreases with decreasing data transfers and is close to zero for highest intensity code.

Validation

To validate this, we will derive the power model parameters W_i for highest intensity purely core-bound code from microscopic level measurements. For simplicity we assume that $W_1 = 0$ on "phinally" system. The only dynamic energy we burn is 830 pJ/Flop. At peak performance 21.6 GF/s, the dynamic power consumption per core is $W_d = W_2 f^2$ = 830 pJ/F * 21.6 GF/s = 17.9 W. Thus, the quadratic factor W_2 = 17.9/2.7^2 W/GHz^2 =2.45 W/GHz^2 and the power model for highest intensity code at single core W^{CPU}=2.45f^2 W + 15.9 W is in accordance with measurements.

Chapter 9

Conclusion

Power dissipation of microprocessors is becoming a critical issue. Although the power output of a single CPU has been capped for practical reasons at around 100–150 W for some time now, building massively parallel supercomputers becomes harder and harder due to the challenge of getting several tens of megawatts into (and out of) a very small space. Moreover, the cost of running large systems constitutes a significant part of the total ownership cost, at least in countries like Germany where electrical energy comes at a rather large price. In the very near future users of supercomputer facilities may be charged not by CPU-hours alone but by the amount of energy they require for solving their numerical problems. Any strategy that allows the user to reduce the power dissipation and/or energy consumption of their running program will contribute to the overall energy efficiency of the machine.

This thesis has investigated several dimensions of the power dissipation and energy consumption of modern processors on the node level when running scientific (loop-centric) workloads. It started from the assertion that there are two basic types of code with regard to scalability across the cores of a multicore chip: either the performance scales linearly with the number of cores, or there is a bottleneck that leads to performance saturation. There is certainly a large spectrum in between those corner cases, but they reflect clearly the underlying hardware bottlenecks. This is why most benchmark studies were done with two simple codes: a large dense matrix-matrix multiplication for the compute-bound case, and a two-dimensional Jacobi smoother for the memory-bound case. A conjugate gradient (CG) algorithm was considered as a near-application case. It is interesting case here since it may show neither saturating nor scalable behavior, depending on the structure of the sparse matrix.

Using the benchmark codes, a study of the performance, power and energy consumption characteristics of the benchmarks was done in order to connect to published results [10]. In addition to previous work, the power dissipation of the system DRAM was also considered and found to be highly dependent on the memory bandwidth utilization. All measurements were based on Intel's RAPL (Running Average Power Limit) counters on four recent Intel Architectures: Sandy Bridge EP, Ivy Bridge EP (in the RRZE Emmy cluster) and Haswell EP.

Subsequently, the analytical power model from [10] was validated against the benchmarks. It was found that the model needed an extension in order to incorporate the effect of varying baseline power (W_0) with increasing number of active cores. The extended model is able to describe the power dissipation characteristics of the CPUs more accurately, but there is a residual variation of the parameters with core count. A model for memory (DRAM) power dissipation was set up that assumes a power draw linear in the memory bandwidth with a constant baseline

power. This model works well for any specific benchmark code, but it was observed that the model parameters vary considerably with the actual memory access pattern (continuous vs. burst mode, number of concurrent streams). Putting the CPU and memory power models together it was possible to construct a comprehensive power dissipation and, adding a suitable performance model, energy consumption model. The consequences for code execution when aiming at smaller power, smaller energy to solution, or smaller energy-delay product (EDP) were investigated and a comparison between the architectures under consideration was drawn. The Ivy Bridge CPUs in the Emmy cluster constituted a special case in this study: due to their low base clock speed, the power vs. frequency characteristic is dominantly linear. This leads to the conclusion that those CPUs should always be run at their base clock speed (or even turbo mode) for scalable workloads, no matter which target metric one wants to optimize for. On CPUs with a quadratic behaviour, an optimal frequency can be calculated from the model that allows for minimum energy to solution; the energy-delay product is still minimal at highest clock speed.

One aspect of power dissipation that is often neglected is the variation of power characteristics across CPUs. The Emmy cluster at RRZE with its ≈1200 CPU chips constitutes a useful platform for such an investigation. It was found that there is considerable variation in power consumption for both benchmarks across the chips. However, chips within the same node are consistently extremely similar. This may lead to several different conclusions: either the variation is caused by a component that is not part of the CPU but of the motherboard, or the system integration process for the nodes leads to similar (same-lot) CPUs ending up in the same node. The DRAM power dissipation for the compute nodes showed similar variations. The 16 "accelerated" nodes in Emmy are equipped with faster DRAM (1866 MHz instead of 1600 MHz), so it was expected that those should have a higher DRAM power. However, the faster DRAM dissipates much less power, which shows that there are factors influencing DRAM power beyond clock speed, such as the manufacturing process, the actual DIMM organization, etc. Whatever the reason for the variations in CPU and DRAM power, it is possible to leverage this effect for saving power by intelligent job scheduling. Some work in this direction has been done at LRZ Garching [29].

One remaining question to be addressed was the connection between "microscopic" power dissipation parameters such as pJ/flop or pJ/byte and the "macroscopic" quantities in our multicore power model. Along the lines of [14], the general power behaviour of the Sandy Bridge CPU was investigated using a tunable-intensity benchmark. As expected, a peak in power dissipation occurs at the "knee" of the Roofline curve, where arithmetic units and memory hierarchy are both fully utilized. This characteristic is lost as soon as the number of threads is reduced to a point where saturation is not possible any more. As for the DRAM, its power is constant versus intensity below the Roofline knee but falls off quickly beyond it. However, even in the saturated case there is a considerable variation with respect to the number of cores. This leads to the conclusion that the bandwidth is not the only parameter that influences the DRAM power; the access characteristics, specifically the number of different streams in memory, also play a decisive role, which is why a simplistic bandwidth-based DRAM power model remains a rough estimate.

From the intensity benchmark results and power measurements with different working set sizes it was possible to infer the microscopic energy consumption parameters that quantify the energy cost of doing a flop in the core or transferring a byte between memory hierarchy levels. It turned out that for single-core execution the baseline power is the significant contribution. This leads to the important realization that the energy-saving feature of SIMD execution (e.g., using

AVX instead of SSE or scalar instructions) is due to the baseline power, since any reduction in runtime immediately causes an (almost) proportional reduction in energy. This situation changes when multiple cores execute code, since the dynamic power becomes much more prominent in this case.

The microscopic parameters fixed by the micro-benchmarks could be used to derive the macroscopic parameter W_2, which quantifies the power vs. clock speed characteristic. Hence, a successful connection was made between microscopic power parameters and our phenomenological power model.

Bibliography

[1] MA Suleman, MK Qureshi, and YN Patt, "Feedback-driven threading: power-efficient and high-performance execution of multi-threadedworkloads on cmps", *ACM SIGARCH Computer Architecture News*, vol. 36, no. 1, p. 277, Mar. 2008.

[2] R Chandra, R Menon, L Dagum, D Kohr, D Maydan, and J McDonald, *Parallel Programming in OpenMP*. 2000, p. 231.

[3] *Message passing interface forum, http://www.mpi-forum.org/.*

[4] *Top 500list, http://www.top500.org/system/178324.*

[5] *Green 500list, http://www.green500.org/greenlists.*

[6] J Doweck, "Inside intel core microarchitecture and smart memory access", *Access*, p. 12,

[7] A Ali, "Intel 64 and ia-32 architectures optimization reference manual", Tech. Rep. 03, 2005, pp. 1–660.

[8] T Horvath and K Skadron, "Multi-mode energy management for multi-tier server clusters", in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08*, New York, New York, USA: ACM Press, 2008, p. 270.

[9] J Treibig, G Hager, and G Wellein, "Likwid: a lightweight performance-oriented tool suite for x86 multicore environments", in *Proceedings of the International Conference on Parallel Processing Workshops*, IEEE, Sep. 2010, pp. 207–216. arXiv: `1004.4431`.

[10] G Hager, J Treibig, J Habich, and G Wellein, "Exploring performance and power properties of modern multicore chips via simple machine models", *CoRR*, vol. abs/1208.2, pp. 1–23, Jan. 2012. arXiv: `arXiv:1208.2908v4`.

[11] S Williams, A Waterman, and D Patterson, "Roofline: an insightful visual performance model for floating-point programs and multicore architectures", *Communications of the ACM*, vol. 52, no. 4, p. 65, Apr. 2009.

[12] MA Awan and SM Petters, "Enhanced race-to-halt: a leakage-aware energy management approach for dynamic priority systems", in *Proceedings - Euromicro Conference on Real-Time Systems*, 2011, pp. 92–101.

[13] R Ge, X Feng, and KW Cameron, "Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters", in *Proceedings of the ACM/IEEE 2005 Supercomputing Conference, SC'05*, vol. 2005, 2005.

[14] JW Choi, D Bedard, R Fowler, and R Vuduc, "A roofline model of energy", in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IEEE, May 2013, pp. 661–672.

[15] J Demmel, A Gearhart, B Lipshitz, and O Schwartz, "Perfect strong scaling using no additional energy", in *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013*, 2013, pp. 649–660.

[16] S W.keckler, W J.Dally, B Khailany, M Garland, and D Glasco, "Gpus and the future of parallel computing", *IEEE Computer Society*, pp. 7–17, 2011.

[17] *Intel corp. intel xeon processor, http://www.intel.com/.*

[18] R Rajwar, *Going under the hood with intels next generation microarchitecture codename haswell*, 2012.

[19] *Likwid performance tools, https://github.com/rrze-likwid/likwid/wiki.*

[20] D W.Marquardt, "An algorithm for least-squares estimation of nonlinear parameters", *Society for industrial and applied mathematics*, vol. 11, pp. 431–441, Jun. 1963.

[21] M Lampton, "Dampingundamping strategies for the levenbergmarquardt nonlinear least-squares method", *Computers in Physics*, vol. 11, no. 1, p. 110, 1997.

[22] *Matlab lma, http://de.mathworks.com/.*

[23] H Stengel, J Treibig, G Hager, and G Wellein, "Quantifying performance bottlenecks of stencil computations using the execution-cache-memory model", *CoRR*, vol. abs/1410.5, pp. 1–10, 2014. arXiv: 1410.5010.

[24] M Kreutzer, G Hager, and G Wellein, "A unified sparse matrix data format for modern processors with wide simd units", *ArXiv preprint arXiv: ...*, vol. 36, no. 5, pp. 1–25, Jan. 2013. arXiv: 1307.6209.

[25] Wa Wulf and Sa McKee, "Hitting the memory wall", *ACM SIGARCH Computer Architecture News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[26] V Cuppu and B Jacob, "Concurrency, latency, or system overhead: which has the largest impact on uniprocessor dram-system performance?", *Proceedings 28th Annual International Symposium on Computer Architecture*, 2001.

[27] D Hackenberg, R Schöne, T Ilsche, D Molka, J Schuchart, and R Geyer, "An energy efficiency feature survey of the intel haswell processor", *Proc. HPPAC 2015*, 2015.

[28] DW SCOTT, "On optimal and data-based histograms", *Biometrika*, vol. 66, no. 3, pp. 605–610, 1979.

[29] A Auweter, A Bode, M Brehm, and L Brochard, "A case study of energy aware scheduling on supermuc", *Supercomputing*, pp. 394–409, 2014.

[30] HT Kung, "Memory requirements for balanced computer architectures", *ACM SIGARCH Computer Architecture News*, vol. 14, no. 2, pp. 49–54, Jun. 1986.

[31] D Callahan, J Cocke, and K Kennedy, "Estimating interlock and improving balance for pipelined architectures", *Journal of Parallel and Distributed Computing*, vol. 5, no. 4, pp. 334–358, Aug. 1988.

[32] RW Hockney and IJ Curington, ": a parameter to characterize memory and communication bottlenecks", *Parallel Computing*, vol. 10, no. 3, pp. 277–286, May 1989.

[33] W Schönauer, *Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers*. Self-edition, 2000.

[34] S Carr and K Kennedy, "Improving the ratio of memory operations to floating-point operations in loops", *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 6, pp. 1768–1810, Nov. 1994.

[35] T Roehl, J Treibig, G Hager, and G Wellein, "Overhead analysis of performance counter measurements", in *2014 43rd International Conference on Parallel Processing Workshops*, IEEE, pp. 176–185.

[36] J Treibig and G Hager, "Introducing a performance model for bandwidth-limited loop kernels", in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6067 LNCS, 2010, pp. 615–624. arXiv: 0905.0792.

Appendix A

List of Symbols and Abbreviations

Symbol	Description
f_0	Baseline clock speed
Δf	Clock speed change
$1 + \Delta\nu$	Normalized clock speed
f_{opt}	Optimal frequency
n_s	Number of cores at saturation point
kWh	Kilowatts hours
P_0	Serial code performance
P_{max}	Maximum performance
T	Total execution time of a program
I	Identity matrix
J	Jacobian matrix
λ	Damping factor
B, b_s	Memory bandwidth, Saturated memory bandwidth
B_m, B_c	Machine balance, Code balance
I_m, I_c	Machine intensity, Computational Intensity
N_{nnz}, N_{nzr}	Number of non-zeros, Number of non-zeros per row
W_0	Baseline power dissipation
W_{00}	Fixed part of baseline power dissipation
w_{01}	Variable part of baseline power dissipation
w_1	Linear part of dynamic power dissipation
w_2	Quadratic part of dynamic power dissipation
W_0^{DRAM}	Fixed background DRAM power
W^{CPU}, E^{CPU}	Dynamic CPU power dissipation, CPU energy to solution
W^{DRAM}, E^{DRAM}	DRAM power dissipation, DRAM energy to solution
W^{TOTAL}, E^{TOTAL}	Total dynamic power dissipation, Total energy to solution
S0, S1	Socket 0, Socket 1
h	Histogram
b	Number of bins
w	Bin width
σ	Standard deviation
ε_{byte}	Energy per byte

ε_{flop}	Energy per flop

Abbreviations	Description
CPU	Central Processing Unit
DRAM	Dynamic Random Access Memory
MPI	Message Passing Interface
ISA	Instruction Set Architecture
CPI	Cycles Per Instruction
IPC	Instruction Per Cycle
Flops	Floating point operations
PUE	Power Utility Efficiency
SMT	Simultaneous Multi-Threading
DVFS	Dynamic Voltage Frequency Scaling
DCT	Dynamic Concurrency Throttling
HPC	High Performance Computing
RAPL	Running Average Power Limit
TDP	Thermal Design power
AVX	Advanced Vector Extensions
FMA	Fused Multiply Add
DP	Double Precision
SP	Single Precision
FP	Floating Point
BIOS	Basic Input/Output System
NEC	Nippon Electric Company
SIMD	Single Instruction Multiple Data
DDR	Double Data Rate
QDR	Quad Data Rate
DIMM	Dual In-line Memory Module
LM	Levenberg-Marquardt
GN	Gauss-Newton
CG	Conjugate Gradient
DGEMM	Dense matrix-matrix multiplication
LUP	Lattice site Updates
spMVM	Sparse Matrix-Vector Multiplication
SDRAM	Synchronous Dynamic Random Access Memory
DDR	Double Data Rate SDRAM
EDP	Energy Delay Product
ED^2P	Energy Delay square Product
ED^3P	Energy Delay cube Product
pdf	Probability Density Function
SD	Standard Deviation
CL	Cache Line
LD/ST	Load and/or Store
ECM	Execution Cache Memory

Appendix B

Matlab codes

B.1. Non-linear Curve fitting Levenberg-Marquardt Algorithm

```
function [fitresult, gof] = createFit(f, w)
[xData, yData] = prepareCurveData( f, w );

% Set up fittype and options.
ft = fittype( 'a*x^2+b*x+c', 'independent', 'x', 'dependent', 'y' );
opts = fitoptions( ft );
opts.Algorithm = 'Levenberg-Marquardt';
opts.Display = 'Off';
opts.Lower = [-Inf -Inf -Inf];
opts.StartPoint = [x y z];
opts.Upper = [Inf Inf Inf];

% Fit model to data.
[fitresult, gof] = fit( xData, yData, ft, opts );

% Create a figure and Plot fit with data.
figure( 'Name', ' nonlinear least-squares fit ' );
subplot( 2, 1, 1 );
h = plot( fitresult, xData, yData );
legend( h, 'w vs. f', ' nonlinear least-squares fit ');
xlabel( 'f' );
ylabel( 'w' );
grid on

% Plot residuals.
subplot( 2, 1, 2 );
h = plot( fitresult, xData, yData, 'residuals' );
legend( h, ' nonlinear least-squares fit  residuals');
xlabel( 'f' );
ylabel( 'w' );
grid on
```

B.2. Histogram Plot Matlab code

```
function pd1 = createFit(x)
x = x(:);
clf;
hold on;

% --- Plot histogram data
[CdfF,CdfX] = ecdf(x,'Function','cdf');
BinInfo.rule = 2;
[~,BinEdge] = internal.stats.histbins(x,[],[],BinInfo,CdfF,CdfX);
[BinHeight,BinCenter] = ecdfhist(CdfF,CdfX,'edges',BinEdge);
hLine = bar(BinCenter,BinHeight,'hist');
set(hLine,'FaceColor','none','EdgeColor',[0.333333 0 0.666667],...
'LineStyle','-', 'LineWidth',1);
xlabel('Data');
ylabel('Density')

% --- Create grid where function will be computed
XLim = get(gca,'XLim');
XLim = XLim + [-1 1] * 0.01 * diff(XLim);
XGrid = linspace(XLim(1),XLim(2),100);

% --- Create probability density function fit
pd = fitdist(x, 'normal');
YPlot = pdf(pd,XGrid);
hLine = plot(XGrid,YPlot,'Color',[1 0 0],...
'LineStyle','-', 'LineWidth',2,...
'Marker','none', 'MarkerSize',6);

% Adjust figure
box on;
grid on;
hold off;
```

Appendix C

C++/Fortran codes

C.1. Dense matrix-matrix multiplication

```
program benchmark
implicit none
integer DIM , ITER
integer i, j , k, it
real(KIND=8), allocatable, dimension(:,:) :: A, B, C
real*8 wcTime, wcTimeStop, wcTimeStart
real*8 cpuTime, cpuTimeStop, cpuTimeStart
real*8 flops_wc, flops_cpu

write(*,*) " Matrix dimension ? # Iterations   ?"
read(*,*) DIM , ITER
! matrix dimension and number of iterations
write(*,*) " Matrix dimension = ",DIM
write(*,*) " # Iterations    = ",ITER

allocate (A(DIM,DIM) )
allocate (B(DIM,DIM) )
allocate (C(DIM,DIM) )

do i = 1 , DIM
do j = 1 , DIM
A(i,j) = dble(i*j)
B(i,j) = dble(i-j)
C(i,j) = 0.d0
enddo
enddo

! Now MKL`s DGEMM as reference
call timing( wcTimeStart , cpuTimeStart ) ! start clock
   do it = 1, ITER
      call DGEMM('n','n',DIM,DIM,DIM,1.d0,A,DIM,B,DIM,1.0,C,DIM)
      call dummy(A(DIM/2,DIM/2))
   enddo
call timing( wcTimeStop , cpuTimeStop ) ! end time
wcTime = wcTimeStop - wcTimeStart
cpuTime=cpuTimeStop -cpuTimeStart
flops_wc = 2.d0 * dble(ITER) * dble(DIM) * dble(DIM) * dble(DIM) / ( wcTime * ↩
    1000000.d0)
flops_cpu = 2.d0 * dble(ITER) * dble(DIM) * dble(DIM) * dble(DIM) / ( cpuTime * ↩
    1000000.d0)
write(*,'(2(a,g20.12),a,g20.12,a)') " MKL [MFlop/s] = ", flops_wc , "WCTime = ",↩
    wctime
end
```

dummy.c

```
void dummy_( double *a , double *b, double *c)
{
}
void dummy( double *a , double *b, double *c)
{
}
```

Makefile for Intel9.1 compiler

```
BHOME =$(PWD)

F90   = ifort
CC = icc

LOWOPT   =  -O2 -xAVX
COPTS =
FLONG = -132

HIOPT =  -O2 -xAVX
NOLINK= -c
LINKF = -mkl

MM.exe: matstuff.F timing.o dummy.o
$(F90) -DSTANDARD $(FLONG) $(HIOPT) $(NOLINK) matstuff.F
$(F90) -o MM.exe matstuff.o timing.o dummy.o $(LINKF)

.c.o:
$(CC) $(COPTS) $(NOLINK) $<

clean:
rm -f *.o *.exe
```

C.2. 2D Openmp jacobi stencil code

```
#include <stdio.h>
#include <likwid.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/resource.h>

#ifdef _OPENMP
#include <omp.h>
#endif

#include "relax_line.h"
#define min(a, b) (((a) < (b)) ? (a) : (b))
#define max(a, b) (((a) > (b)) ? (a) : (b))

void timing(double* wcTime, double* cpuTime){
   struct timeval tp;
   struct rusage ruse;
   gettimeofday(&tp, NULL);
   *wcTime=(double) (tp.tv_sec + tp.tv_usec/1000000.0);
   getrusage(RUSAGE_SELF, &ruse);
   *cpuTime=(double)(ruse.ru_utime.tv_sec+ruse.ru_utime.tv_usec / 1000000.0);
```

```
}

int main(int argc, char *argv[]) {
LIKWID_MARKER_INIT;
#pragma omp parallel{
   LIKWID_MARKER_THREADINIT; }
if (argc != 6){
   printf("Usage: Please provide isize and jsize including boundary layer.\n");
   printf("Usage: %s isize jsize iblocksize|0 jblocksize|0 sweeps\n", argv[0]);
   return -1;
}
int is=atoi(argv[1]);
int js=atoi(argv[2]);
int iblocksize=atoi(argv[3]);
if (iblocksize>is) iblocksize = is;
if (iblocksize==0) iblocksize = is;
int jblocksize=atoi(argv[4]);
if (jblocksize>js) jblocksize = js;
if (jblocksize==0) jblocksize = js;
int sweeps=atoi(argv[5]);

int Ni=is;
if (Ni%2==0) Ni+=1; !odd line length for sse
int Nj=js;
double arrayskb = Ni * Nj * 2.0 * 8.0 / 1000.0;  ! memory usage [kB] of both ↩
    arrays
double min_runtime;
double mlups;
double wct_start,wct_end,cput_start,cput_end;
const int grid_size = Ni * Nj;
const int line_size = Ni;
double * tmp_grid;
int align_to = 16;
int ok;
double * dg;
ok = posix_memalign((void**)&dg, align_to, (grid_size)*sizeof(double));
if(ok != 0) return -1;

! align other layer
double * sg;
ok = posix_memalign((void**)&sg, align_to,(grid_size+1)*sizeof(double));
if(ok != 0) return -1;

double * source_grid = &sg[1];
double * destination_grid = dg;
int a;
for (a=0; a<grid_size; a++){
   destination_grid[a] = 1.0;
   source_grid[a] = 3.0;
   }
timing(&wct_start, &cput_start);
int l;
for (l=0; l<sweeps; ++l){
  #pragma omp parallel
   {
      LIKWID_MARKER_START("J2D");
   }
   int i, j;
   int iblock, jblock;
   int istart, jstart, iend, jend;

   for(jblock=1; jblock<Nj-1; jblock+=jblocksize){
      for(iblock=1; iblock<Ni-1; iblock+=iblocksize){
      jstart = jblock;
      jend = min(jblock+jblocksize-1, Nj-2);
```

```
87      #pragma omp parallel for schedule (static)
88         for(j=jstart; j<=jend; ++j){
89         istart = iblock;
90         iend = min(iblock+iblocksize-1, Ni-2);
91         relax_line( istart, iend, &destination_grid[j*line_size], &source_grid[↩
               j*line_size], &source_grid[(j-1)*line_size], &source_grid[(j+1)*↩
               line_size] );
92         } ! j
93      } ! iblocks
94   } ! jblocks
95   #pragma omp parallel{
96      LIKWID_MARKER_STOP("J2D"); }
97   tmp_grid = destination_grid; destination_grid = source_grid; source_grid = ↩
        tmp_grid;
98 } ! sweeps
99
100 ! prevent compiler from eliminating loop
101 if (source_grid[22]<0) printf("\n argh %f\n", destination_grid[22]);
102 timing(&wct_end, &cput_end);
103 min_runtime = wct_end-wct_start;
104 free(sg);
105 free(dg);
106 mlups = ((double)sweeps * (double)(Ni-2) * (double)(Nj-2)) / (double)min_runtime↩
      / 1000000.0;
107 printf("sizeofbotharrays[kB] %f runtime %f sweeps %i isize %i jsize %i iblock %i↩
      jblock %i mlups %f \n", arrayskb, min_runtime, sweeps, is, js, iblocksize, ↩
     jblocksize, mlups);
108 LIKWID_MARKER_CLOSE;
109 return 0;
110 }
```

relax_line.c

```
1  void relax_line( int istart, int iend, double *restrict dest_line, double *↩
       restrict source_line, double *restrict source_above, double *restrict ↩
       source_under ){
2     int start = istart;
3     !  peel off one unaligned iteration if necessary (16-byte alignment)
4     if (((long)(&dest_line[start]) & 0x0F) != 0){
5        dest_line[start]=( source_above[start] + source_under[start] + source_line↩
             [start-1] + source_line[start+1] )*0.25;
6        start++;
7     }
8     int i;
9     #pragma vector aligned
10    #pragma simd
11    for(i=start; i<=iend; i++){
12       dest_line[i]=( source_above[i] + source_under[i] + source_line[i-1] + ↩
             source_line[i+1] )*0.25;
13    }
14 }
```

C.3. OpenMP-parallel Conjugate Gradient Method

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include <omp.h>
4 #include "mmio.h"
5 #include <sys/time.h>
```

```
#include <likwid.h>
int M, N, nz;
double conj_grad(int colidx[], int rowstr[], double b[], double x[], double a[],↩
     double p[], double q[],double r[], double *rnorm);

int main(int argc, char **argv) {
int i,j, k, it;
int nthreads = 1;
double rnorm;
double t;
double wct_start, cput_start, wct_end, cput_end;
int *colidx, *rowstr;
double *a, *b, *x, *p, *q, *w, *r;
LIKWID_MARKER_INIT;

%*********  Reading sparse matrix   ***************/
int ret_code;
MM_typecode matcode;
FILE *f;
int *I, *J, *temp;
double *val;

if (argc < 2){
   fprintf(stderr, "Usage: %s [martix-market-filename]\n", argv[0]);
   exit(1);
   }
   else{
   if ((f = fopen(argv[1], "r")) == NULL)
   exit(1);
}

if (mm_read_banner(f, &matcode) != 0){
   printf("Could not process Matrix Market banner.\n");
   exit(1);
}

if (mm_is_complex(matcode) && mm_is_matrix(matcode) && mm_is_sparse(matcode)){
   printf("Sorry, this application does not support ");
   printf("Market Market type: [%s]\n", mm_typecode_to_str(matcode));
   exit(1);
}

!... find out size of sparse matrix ....
if ((ret_code = mm_read_mtx_crd_size(f, &M, &N, &nz)) != 0)
   exit(1);

!... reseve memory for matrices
I = (int *)malloc(nz * sizeof(int));
J = (int *)malloc(nz * sizeof(int));
val = (double *)malloc(nz * sizeof(double));
! reading in doubles, ANSI C requires the use of the "l" as "%lg", "%lf", "%le"
for (i = 0; i < nz; i++){
   fscanf(f, "%d %d %lg\n", &I[i], &J[i], &val[i]);
   I[i]--;  !... adjust from 1-based to 0-based
   J[i]--;
}
if (f != stdin) fclose(f);

%************  Write out matrix  ************/
mm_write_banner(stdout, matcode);
mm_write_mtx_crd_size(stdout, M, N, nz);
for (i = 0; i < nz; i++)
   fprintf(stdout, "%d %d %20.19g\n", I[i] + 1, J[i] + 1, val[i]);

printf("\n\n CG Benchmark\n");
```

```
printf(" Size: %d * %d\n", M, N);
printf(" Number of non-zeros: %d\n", nz);
colidx = (int *)malloc(nz * sizeof(int));
rowstr = (int *)malloc((M + 1) * sizeof(int));
a = (double *)malloc(nz * sizeof(double));
b = (double *)malloc(N  * sizeof(double));
x = (double *)malloc(N  * sizeof(double));
p = (double *)malloc(N  * sizeof(double));
q = (double *)malloc(N  * sizeof(double));
r = (double *)malloc(N  * sizeof(double));
w = (double *)malloc(N  * sizeof(double));

%*************** Store matrix in CRS format *****************/
int   p1;
rowstr[0] = 0;
int nzr = 0;
for (p1 = 0; p1 <M; p1++){
   for (i = 0; i < nz; i++){
      if (I[i] == p1){
         a[nzr] = val[i];
         colidx[nzr] = J[i];
         nzr++;
      }
   }
   rowstr[p1 + 1] = nzr;
}

%*****************  Printing matrix in CRS format ***************/
printf("\nCRS Format\n");
for (i = 0; i <nz; i++){
   printf( "colidx %d \n", colidx[i]+1);
}
for (j = 0; j <nz; j++){
   printf("value %20.19g\n", a[j]);
}
for (k = 0; k < M +1; k++){
   printf("rowstr: %d \n", rowstr[k]+1);
}

srand(time(NULL));
#pragma omp parallel for private(i)
for (i = 0; i < N; i++) {
   b[i] = (double)(rand() % 10);
}

%*************    BenchMark   *****************/
printf("\nBenchmark start\n");
#pragma omp parallel private(k){
   LIKWID_MARKER_THREADINIT;
   k = omp_get_num_threads();
   printf("Number of Threads requested = %i\n", k);
}

timing(&wct_start, &cput_start); !... Start of time section
for (it = 1; it <= 1; it++) {
   conj_grad(colidx, rowstr, b, x, a, p, q, r, &rnorm);
}
timing(&wct_end, &cput_end);     !... End of timed section
t = wct_end - wct_start;
printf("    RunTime: %20.14e\n ", t);
printf(" \nBenchmark completed\n\n");
LIKWID_MARKER_CLOSE;
return 0;
}

```

```
%***********   Algorithm: Method of Conjugate Gradients   ******************/
double conj_grad(int colidx[], int rowstr[], double b[], double x[], double a[],↩
     double p[], double q[], double r[], double *rnorm){
double d, sum, rho,rho0, alpha, beta, s;
int k, j;
double eps = 1.0e-10;
int cgit, cgitmax = 90000000;
rho = 0.0;
sum = 0.0;
#pragma omp parallel default (shared) private(j,k,cgit,alpha,beta,s){
#pragma omp for
for (j = 0; j < M; j++) {
   q[j] = 0.0;     !... initionlization
   x[j] = 0.0;     !... Given an initial guess x0
   r[j] = b[j];    !... compute r0
   p[j] = r[j];    !... set p0 = r0
}

!rho: norm of r0. To obtain norm of r: First, sum squares of r elements locally
#pragma omp for reduction(+:rho)
for (j = 0; j < N; j++){
   rho = rho + r[j] * r[j];
}

!...first test for stop
   if (rho > eps){
      for (cgit = 1; cgit <= cgitmax; cgit++){
      #pragma omp single {
         rho0 = rho;  !... Save a temporary of rho
         d = 0.0;
         rho = 0.0;
      } !... end single

      LIKWID_MARKER_START("SPMVM");
      !... Step1: Compute matrix-vector multiply A.p and store it into q
      #pragma omp for schedule(static)
      for (j = 0; j < M; j++) {
         s = 0.0;
         for (k = rowstr[j]; k < rowstr[j + 1]; k++) {
            s = s + a[k] * p[colidx[k]];
         }
         q[j] = s;
      }
      LIKWID_MARKER_STOP("SPMVM");

      !... Step2: Compute <pk, Apk> = <p, q>. Obtain p.q
      LIKWID_MARKER_START("ScalarProduct");
      #pragma omp for reduction(+:d)
      for (j = 0; j < N; j++) {
         d = d + p[j] * q[j];
      }
      LIKWID_MARKER_STOP("ScalarProduct");

      !... Step3: Compute alpha = <rk,rk>/<pk,Apk>= rho/<p,q>
      alpha = rho0 / d;

      !... Step4,5,6: Compute xk + 1 = xk + akpk   and    rk + 1 = rkakApk and <↩
           rk + 1, rk + 1 > Now, obtain the norm of r:
      LIKWID_MARKER_START("computestep456");
      #pragma omp for reduction(+:rho)
      for (j = 0; j < N; j++) {
         x[j] = x[j] + alpha*p[j];
         r[j] = r[j] - alpha*q[j];
         rho = rho + r[j] * r[j];
      }
```

```
      LIKWID_MARKER_STOP("computestep456");

      if (rho < eps){
         break;
      }
      !...  Step7: Compute beta = <rk+1,rk+1>/<rk,rk> = rho / rho0
      beta = rho / rho0;

      !... Step8: Compute pk+1 = rk+1+bkpk
      LIKWID_MARKER_START("computestep8");
      #pragma omp for schedule(static)
      for (j = 0; j < N; j++) {
         p[j] = r[j] + beta*p[j];
      }
      LIKWID_MARKER_STOP("computestep8");
   }  !... end if
} !... end of do cgit=1,cgitmax
printf("Number of Max Iterations = %i\n", cgit-1);

!... Compute residual norm || r || = || b - A.x ||
LIKWID_MARKER_START("postcompute");
#pragma omp for schedule(static)
for (j = 0; j < M; j++) {
   s = 0.0;
   for (k = rowstr[j]; k <= rowstr[j + 1] - 1; k++) {
      s = s + a[k] * x[colidx[k]];
   }
   r[j] = s;
}
LIKWID_MARKER_STOP("postcompute");

LIKWID_MARKER_START("lastcompute");
#pragma omp critical
for (j = 0; j < N; j++) {
   s = b[j] - r[j];
   sum = sum + s*s;}
   LIKWID_MARKER_STOP("lastcompute");
} !... End parellel
(*rnorm) = sqrt(sum);
return 0;
}
```

C.4. Variable Intensity Benchmarks

0.5 F/B Intensity AVX Benchmark (Memory-bound case)

```
STREAMS 1
TYPE DOUBLE
FLOPS 4
BYTES 8
vmovaps ymm5, [SCALAR]
vmovaps ymm6, [SCALAR]
vmovaps ymm7, [SCALAR]
vmovaps ymm8, [SCALAR]
LOOP 16
vmovaps     ymm1, [STR0 + GPR1*8]
vmovaps     ymm2, [STR0 + GPR1*8+32]
vmovaps     ymm3, [STR0 + GPR1*8+64]
vmovaps     ymm4, [STR0 + GPR1*8+96]

vmulpd      ymm1, ymm5, ymm5
vaddpd      ymm1, ymm6, ymm6
```

```
vmulpd      ymm2, ymm5, ymm5
vaddpd      ymm2, ymm6, ymm6
vmulpd      ymm3, ymm5, ymm5
vaddpd      ymm3, ymm6, ymm6
vmulpd      ymm4, ymm5, ymm5
vaddpd      ymm4, ymm6, ymm6
vmulpd      ymm1, ymm7, ymm7
vaddpd      ymm1, ymm8, ymm8
vmulpd      ymm2, ymm7, ymm7
vaddpd      ymm2, ymm8, ymm8
vmulpd      ymm3, ymm7, ymm7
vaddpd      ymm3, ymm8, ymm8
vmulpd      ymm4, ymm7, ymm7
vaddpd      ymm4, ymm8, ymm8
```

16 F/B Intensity AVX Benchmark (Compute-bound case)

```
STREAMS 1
TYPE DOUBLE
FLOPS 128
BYTES 8
vmovaps ymm5, [SCALAR]
vmovaps ymm6, [SCALAR]
vmovaps ymm7, [SCALAR]
vmovaps ymm8, [SCALAR]
vmovaps ymm9, [SCALAR]
vmovaps ymm10, [SCALAR]
vmovaps ymm11, [SCALAR]
vmovaps ymm12, [SCALAR]
LOOP 16
vmovaps     ymm1, [STR0 + GPR1*8]
vmovaps     ymm2, [STR0 + GPR1*8+32]
vmovaps     ymm3, [STR0 + GPR1*8+64]
vmovaps     ymm4, [STR0 + GPR1*8+96]

vmulpd      ymm1, ymm5, ymm5
vaddpd      ymm1, ymm6, ymm6
vmulpd      ymm2, ymm5, ymm5
vaddpd      ymm2, ymm6, ymm6
vmulpd      ymm3, ymm5, ymm5
vaddpd      ymm3, ymm6, ymm6
vmulpd      ymm4, ymm5, ymm5
vaddpd      ymm4, ymm6, ymm6
vmulpd      ymm1, ymm7, ymm7
vaddpd      ymm1, ymm8, ymm8
vmulpd      ymm2, ymm7, ymm7
vaddpd      ymm2, ymm8, ymm8
vmulpd      ymm3, ymm7, ymm7
vaddpd      ymm3, ymm8, ymm8
vmulpd      ymm4, ymm7, ymm7
vaddpd      ymm4, ymm8, ymm8

vmulpd      ymm1, ymm9, ymm9
vaddpd      ymm1, ymm10, ymm10
vmulpd      ymm2, ymm9, ymm9
vaddpd      ymm2, ymm10, ymm10
vmulpd      ymm3, ymm9, ymm9
vaddpd      ymm3, ymm10, ymm10
vmulpd      ymm4, ymm9, ymm9
vaddpd      ymm4, ymm10, ymm10
vmulpd      ymm1, ymm11, ymm11
vaddpd      ymm1, ymm12, ymm12
```

```
vmulpd      ymm2, ymm11, ymm11
vaddpd      ymm2, ymm12, ymm12
vmulpd      ymm3, ymm11, ymm11
vaddpd      ymm3, ymm12, ymm12
vmulpd      ymm4, ymm11, ymm11
vaddpd      ymm4, ymm12, ymm12

.
.
.
.
.
.
.

vmulpd      ymm1, ymm9, ymm9
vaddpd      ymm1, ymm10, ymm10
vmulpd      ymm2, ymm9, ymm9
vaddpd      ymm2, ymm10, ymm10
vmulpd      ymm3, ymm9, ymm9
vaddpd      ymm3, ymm10, ymm10
vmulpd      ymm4, ymm9, ymm9
vaddpd      ymm4, ymm10, ymm10
vmulpd      ymm1, ymm11, ymm11
vaddpd      ymm1, ymm12, ymm12
vmulpd      ymm2, ymm11, ymm11
vaddpd      ymm2, ymm12, ymm12
vmulpd      ymm3, ymm11, ymm11
vaddpd      ymm3, ymm12, ymm12
vmulpd      ymm4, ymm11, ymm11
vaddpd      ymm4, ymm12, ymm12
```

Highest Intensity SSE Benchmark (Compute-bound case)

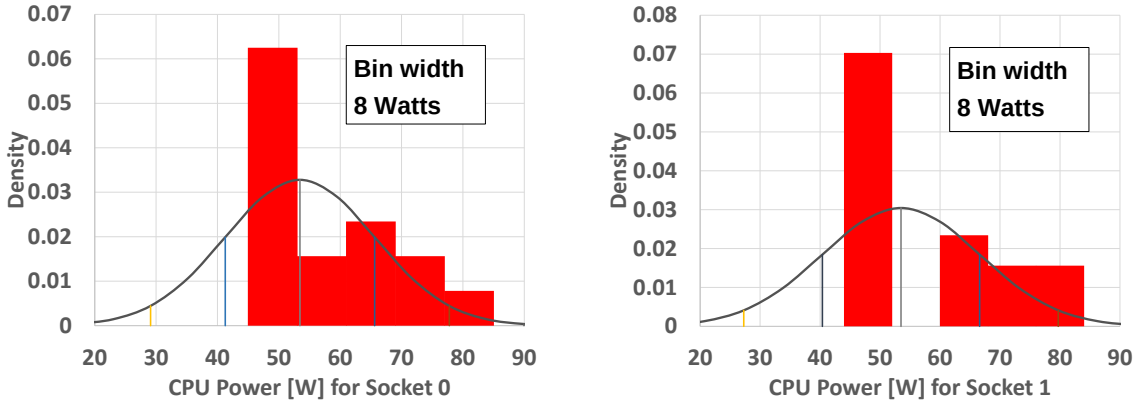
```
STREAMS 0
TYPE DOUBLE
FLOPS 4
BYTES 0
movaps FPR5, [SCALAR]
movaps FPR6, [SCALAR]
movaps FPR7, [SCALAR]
movaps FPR8, [SCALAR]
movaps FPR9, [SCALAR]
LOOP 8

mulpd      FPR5, FPR5
addpd      FPR1, FPR6
mulpd      FPR7, FPR7
addpd      FPR2, FPR6
mulpd      FPR8, FPR8
addpd      FPR3, FPR6
mulpd      FPR9, FPR9
addpd      FPR4, FPR6

mulpd      FPR5, FPR5
addpd      FPR1, FPR6
mulpd      FPR7, FPR7
addpd      FPR2, FPR6
mulpd      FPR8, FPR8
addpd      FPR3, FPR6
mulpd      FPR9, FPR9
addpd      FPR4, FPR6
```

Appendix D

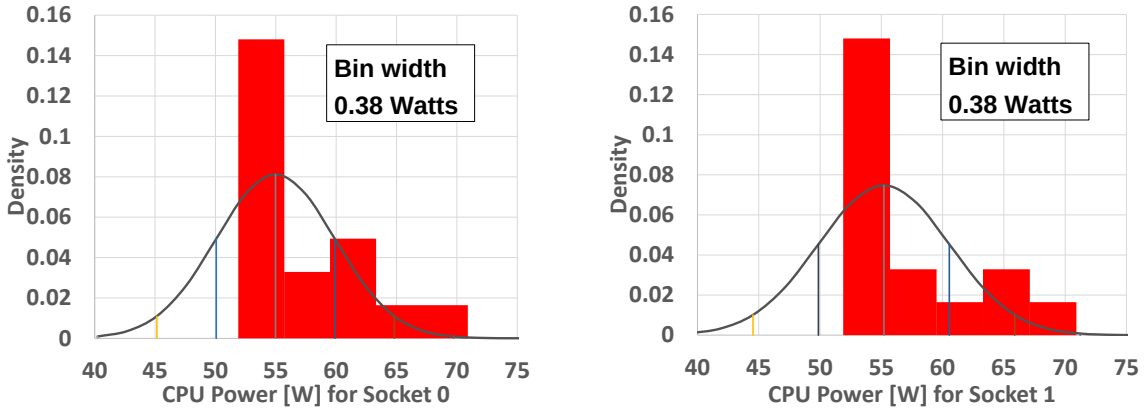
HistogramPlots

D.1. Accelerator Nodes

D.1.1. CPU Power and Model Parameters for DGEMM and Jacobi benchmarks



(a) DGEMM Benchmark Code



(b) Jacobi Benchmark Code

Figure D.1.: Power dissipation by CPU for both DGEMM and Jacobi cases.

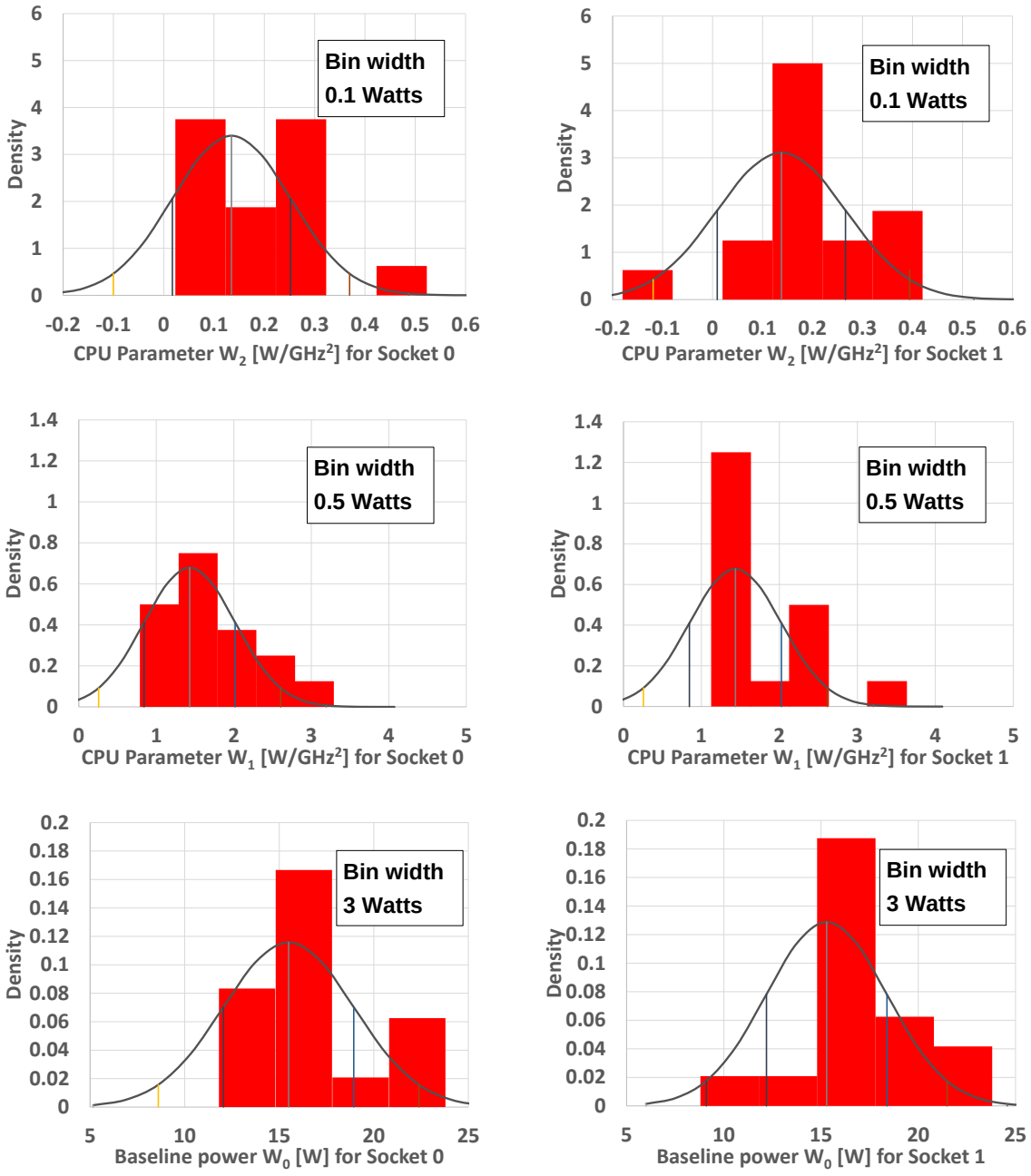


Figure D.2.: DGEMM CPU power parameters W_i .

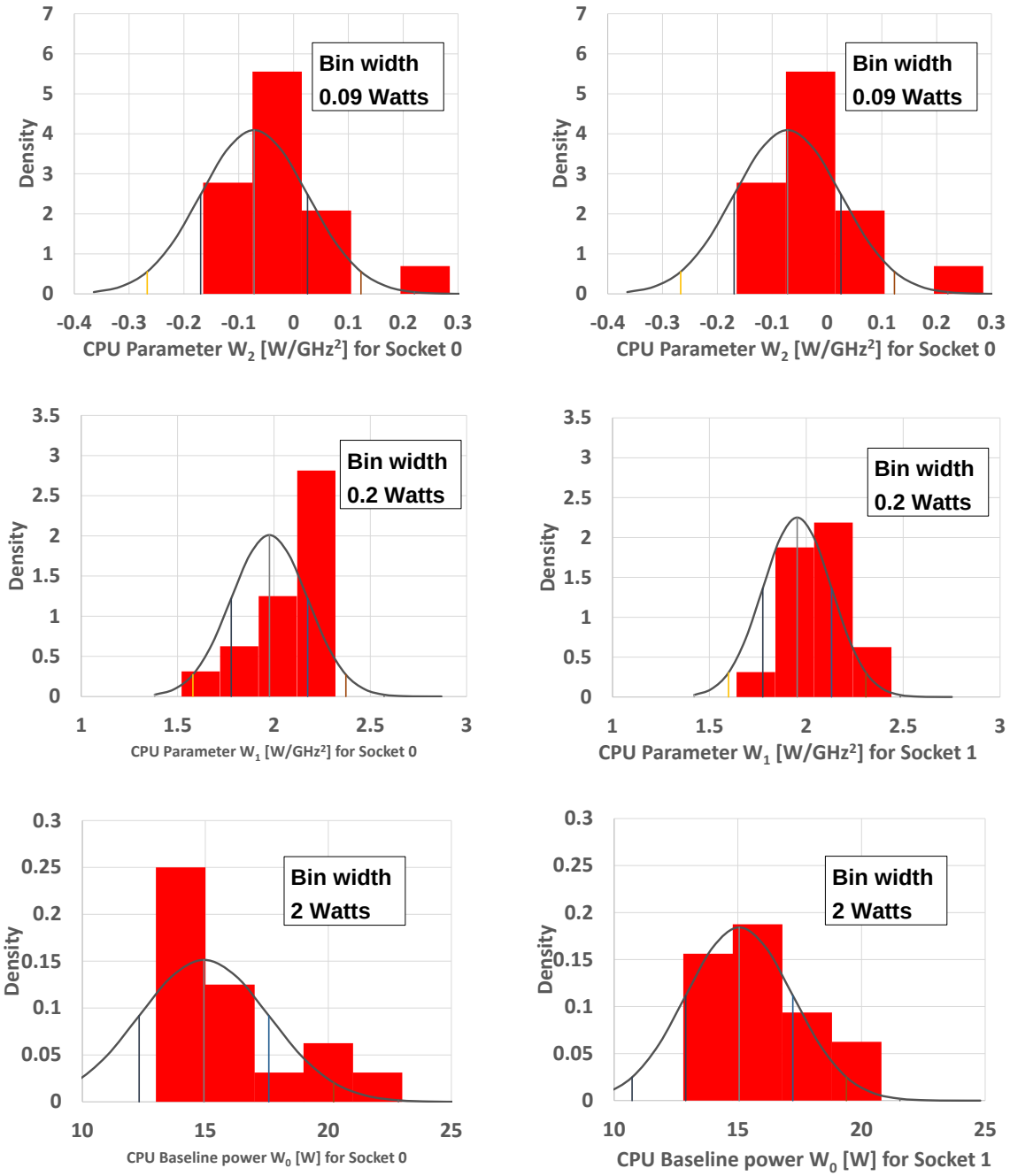


Figure D.3.: Jacobi CPU power parameters W_i .

D.1.2. DRAM Power and Model Parameters for DGEMM and Jacobi benchmarks

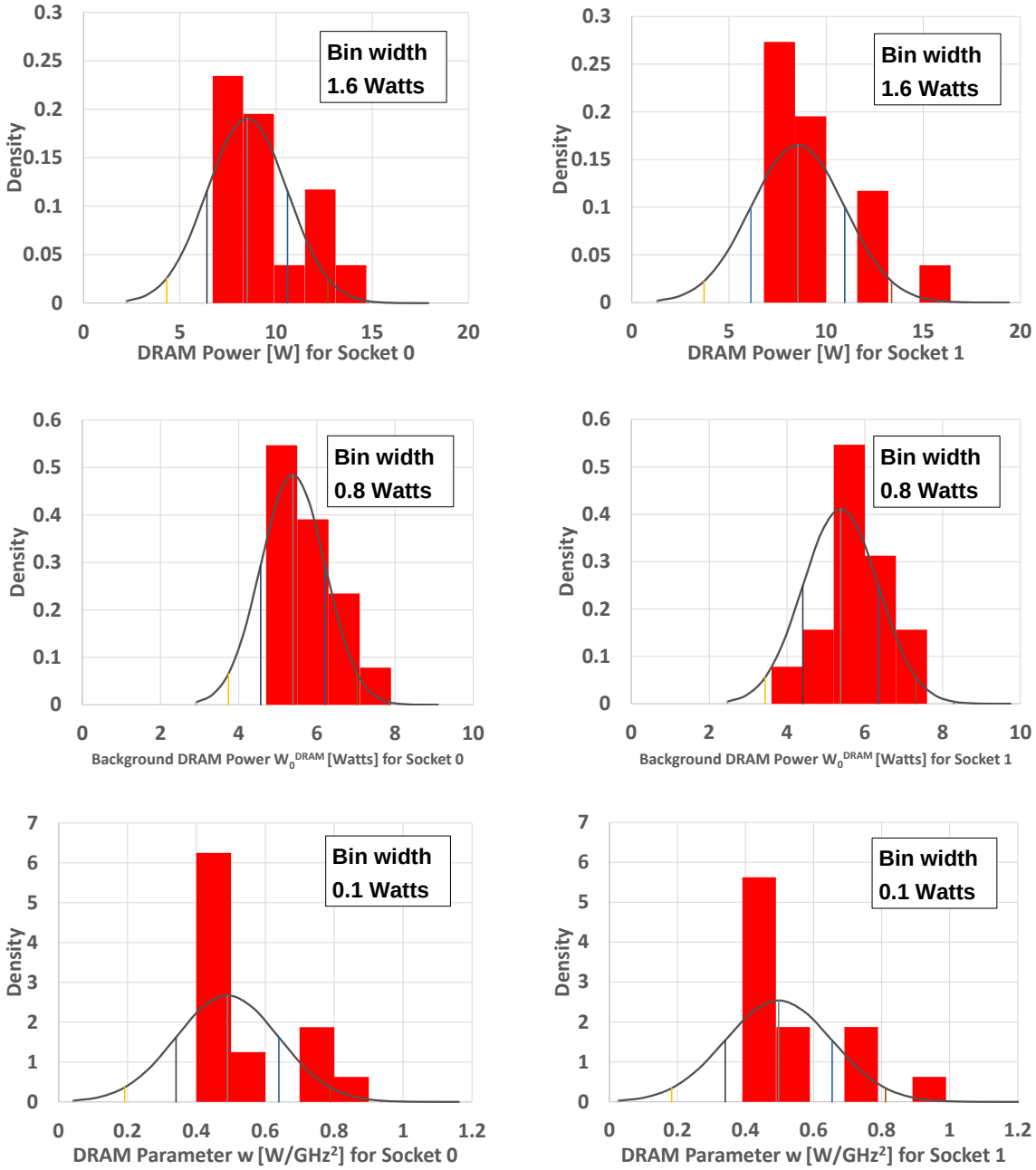


Figure D.4.: DGEMM DRAM power and its model parameters W_i .

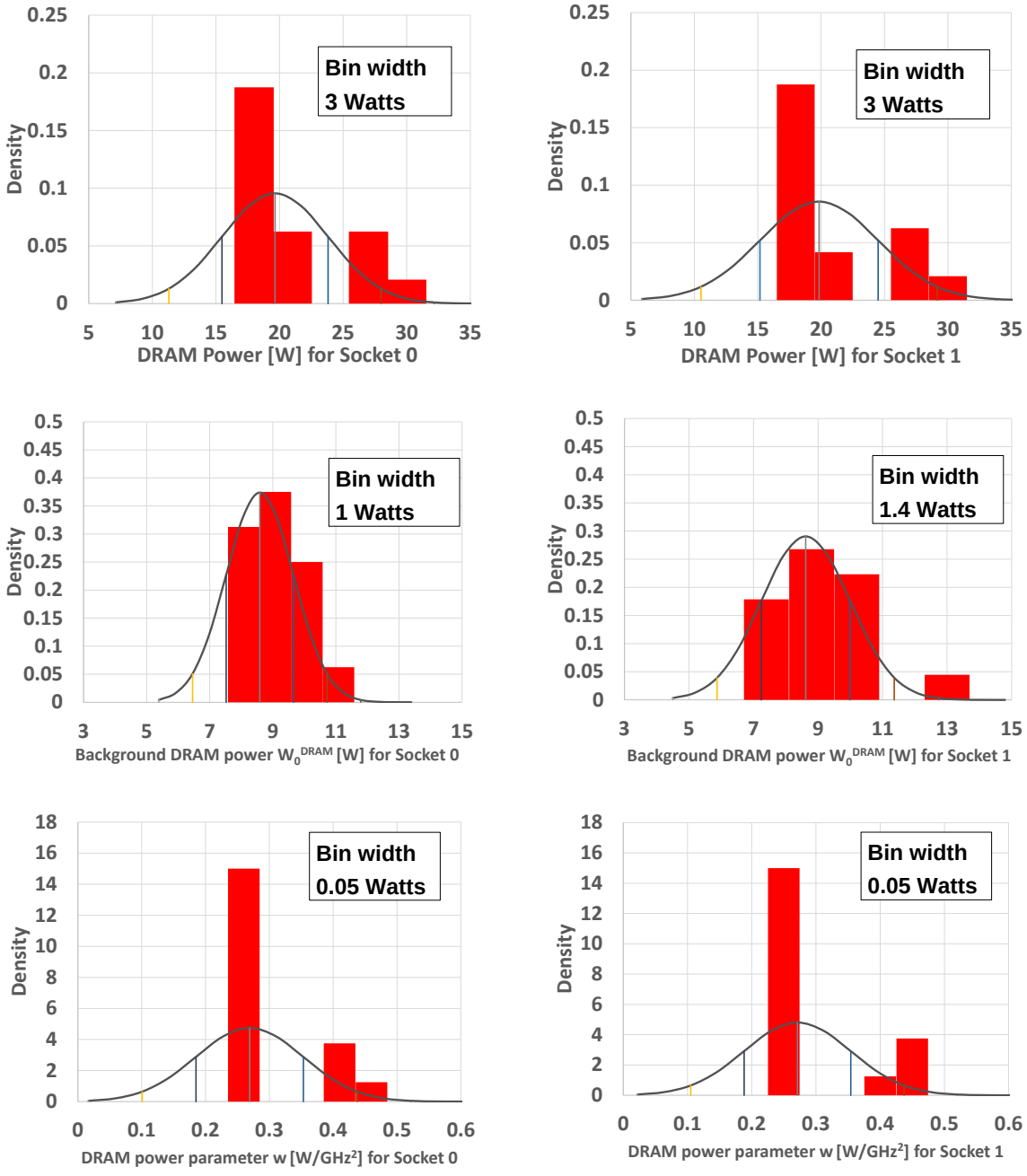


Figure D.5.: Jacobi DRAM power and its model parameters W_i .