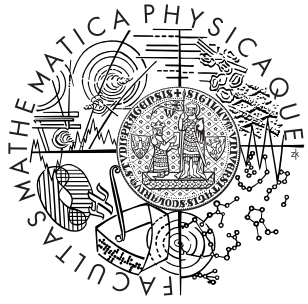
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Tomas Tuma

Efficient gathering of performance information on multicore systems

Department of software engineering

Advisors:
Sean Rooney, Ph.D., IBM Zurich Research Laboratory
Paul Hurley, Ph.D., IBM Zurich Research Laboratory
Petr Tuma, Ph.D., Charles University in Prague

Study program:
Computer Science - Software Systems

2008

Acknowledgements

I would like to thank Dr. Paolo Scotton, the manager of Advanced Messaging Technologies group at IBM Zurich Research Laboratory, for leading my master thesis internship. Big thanks go to Dr. Sean Rooney, the main advisor of the thesis, who carefully led me through the process of exploring the new field, formulating the results and writing the scientific text. I really appreciated his profound approach in our collaboration which significantly contributed to the productivity of my stay at the lab. I would also like to thank Dr. Paul Hurley, who was advising the thesis and with whom we came up with many inventive and extremely interesting ideas in the new field of compressive sampling. Dr. Petr Tuma from the Charles University in Prague helped me with finishing and submitting the thesis.

Parts of the thesis are based on the paper "On the applicability of compressive sampling in fine grained processor performance monitoring" which I wrote together with Sean Rooney and Paul Hurley. The paper was submitted to SIGMETRICS/Performance 2009. Chapter 7 is included in the U.S. patent application Nr. CH9-2008-022 by the same authors, filed in June 2008 by IBM Corporation.

Of course, writing the thesis did not mean only uncountable hours in front of the computer; also the indirect stimulations, be it technical discussions or exploring the nature, were extremely important for me. The whole team of the IBM Zurich Research Laboratory created a friendly, open and collaborative environment which I really enjoyed. Special thanks go to Urs Hunkeler – thanks to him I could explore Switzerland much more than I expected.

I am also grateful to my parents who have supported me throughout my studies.

Rueschlikon, October 30th, 2008

I hereby declare that I wrote the master thesis on my own and using only the cited sources. I agree with lending the thesis.

Prague, November 15th, 2008 Tomas Tuma

Contents

Title: Efficient gathering of performance information on multicore systems
Author: Tomas Tuma
Department: Department of software engineering
Supervisor: Sean Rooney, Ph.D., Paul Hurley, Ph.D., Petr Tuma, Ph.D.
Supervisor's e-mail address: sro@zurich.ibm.com, pah@zurich.ibm.com, petr.tuma@dsrg.mff.cuni.cz
Abstract: Modern multicore processors provide performance counters that export information on various essential aspects of software execution, from instruction decoding to cache utilization. Typically, a processor is capable of counting a small subset from hundreds of different event types, the events themselves can occur almost every processor clock tick. This yields a significant amount of data which is difficult to collect without disrupting the execution itself. The goal of the thesis is to apply compressive sampling - a special method of sampling signals that allows to reconstruct sparse signal from a small number of samples - to the performance counter data.
Keywords: compressive sampling, multicore processors, performance information

Title: Efektivní sběr informací o výkonu na multicore systémech
Author: Tomáš Tůma
Department: Katedra softwarového inženýrství
Supervisor: Sean Rooney, Ph.D., Paul Hurley, Ph.D., Petr Tuma, Ph.D.
Supervisor's e-mail address: sro@zurich.ibm.com, pah@zurich.ibm.com, petr.tuma@dsrg.mff.cuni.cz
Abstract: Moderní vícejádrové (multicore) procesory mají k dispozici registry, prostřednictvím nichž je možné získávat informace o řadě důležitých aspektů výkonnosti systému, od jednotek pro dekódování instrukcí po využití pamětí cache. Procesor obvykle umožňuje sledovat danou podmnožinu ze stovek událostí, které mohou nastávat v každém procesorovém cyklu. Vzniká tak značný objem dat, která je obtížné získávat, aniž by bylo narušeno provádění programu. Cílem diplomové práce je použít kompresní vzorkování - speciální metodu vzorkování signálu, která umožňuje rekonstruovat tzv. řídké signály z relativně malého množství vzorků - na tato data o výkonu procesoru.
Keywords: kompresní vzorkování, vícejádrové procesory, výkonnostní informace

Chapter 1

Introduction

For many years, growing performance requirements on computer processors have been reflected by increasing the number of transistors per integrated circuit and by increasing the processor clock frequency. However, processor design has reached the point where the performance cannot be increased by simple frequency scaling. The reasons can be found in excessive power consumption, heat leakage and space requirements, which combined together make it impossible to improve performance in such a way.

In order to gain better performance, more sophisticated approaches have to be taken. Basically, processor designers can proceed to either qualitative or quantitative changes. Qualitative changes involve changes in the inner processor architecture and instruction set. Whereas the performance gain is rather moderate, costs of such changes are enormous. Quantitative approaches are often simpler to implement on the processor platform, but care has to be taken that the desired performance gain is not significantly reduced by contention on shared resources and that existing applications are able to make use of the new performance capacity. The most wide spread quantitative techniques are simultaneous multithreading (SMT, also known as hyperthreading), symmetric multi-processing (SMP) and chip multi-processing (CMP, also known as multicore processing).

Multicore chip design has turned out to be a promising way of improving chip performance. It is believed that in the following years, multicore architectures with tens to hundreds of cores on one chip will appear. The computer community now faces the challenge of managing the computational power provided by multicore processors. Apart from hardware design issues, multicore chips require software awareness. Consequently, it is reasonable to expect an active development of software that may also lead to updates in the hardware requirements.

1.1 Motivation

Providing a feedback to the software layers is often of key importance to enable an efficient usage of the underlying hardware. When running on a hardware platform with a highly developed infrastructure (such as hyperthreaded, multicore or multiprocessor systems), it is useful to observe performance of the particular system components. This need has been reflected in many modern processor families by adding the support for real-time performance monitoring.

Performance data can be used either to perform a static analysis of critical parts of software or it can be used dynamically to control the system behaviour during the software execution. The latter involves mainly operating system schedulers. To fulfill contradictory requirements, modern schedulers adapt to the current workload characteristics in order to provide a heuristic solution to the scheduling problem. Performance of the operating system scheduler influences the overall system performance.

On multicore and multiprocessor systems, delivering the performance data to the consumer can be a challenge. As the number of performance sources and performance characteristics grows, the bus bandwidth required to transfer the data becomes significant. However, auxiliary data should always be delivered with minor impacts on the overall system performance.

This makes it desirable to apply an on-the-fly compression method to the performance signals. Such a method has to exhibit special properties required by this particular use case. The most important are

good compression rates for the domain of processor performance signals and applicability with minor resource requirements.

1.2 Goal

Compressive sampling [1, 2, 3] is an emerging signal sampling and reconstruction paradigm. Assuming there are certain assumptions met, it guarantees that a signal can be sampled directly in its compressed form, transferred and recovered accurately with a very high probability. The sampling and compression happen in the same phase and are designed to be non-adaptive, i.e. the algorithm does not take different execution paths for different input data. Emphasis is placed on the reconstruction phase, where the signal is estimated using optimization techniques (e.g. linear programming, greedy algorithms for sparse approximation). This asymmetry is useful in applications where simplicity of the sampling phase can be traded against a more complex recovery phase, such as in the transfer of performance information from processor cores to a scheduler.

The goal of this work is to evaluate compressive sampling as a tool for compression of performance signals. This includes verifying whether the assumptions of compressive sampling can be met in the case of performance monitoring, comparison of basic compressive sampling variations and tackling the implementation issues in multicore processors.

1.3 Structure of the document

The remainder of the text is organized as follows.

In Chapter 2, an overview of the related work is given. In particular, the areas of contemporary processor design, performance monitoring and operating system scheduling are visited.

In Chapter 3, the background of compressive sampling is described in a greater detail. First, we begin with the tenet notion of sparsity. Assuming the sparsity of the signal, different theorems available in the literature are presented and categorized. An overview of the algorithms for recovery together with estimates of the time complexity is given.

In Chapter 4, a simple proof of the incoherence between the noiselet and Haar bases is given. Unlike the proof currently available in the literature, the incoherence is shown in the language of simple linear algebra. An elegant recursive equation for noiselets is derived which allows for a Kronecker-product based proof.

In Chapter 5, compressibility of processor performance signals is examined. The hardware and software environment of the experimental setup is described and means of sparsity estimation are introduced. Then, the sparsity of a set of real world signals is estimated.

In Chapter 6, the intrinsic structure of compressive sampling is decomposed into modules and an experimental environment in the *Matlab* software is developed. Using the experimental environment, compressive sampling is applied to the set of real-world performance signals. The results are discussed and possible refinements are suggested.

In Chapter 7, a novel scheme for a per-core compressive sampling module is given. The scheme suggests how a universal compressive sampling module could be implemented with minimal resource requirements.

In Chapter 8, we shortly summarize the obtained results and formulate the conclusions.

1.4 Contributions to the text

This master thesis was written at IBM Zurich Research Laboratory, Switzerland. As such, it originated from the cooperation with my advisors and parts of the text are based on our joint work. The overwhelming majority of the text, Chapters 1 – 5, 6.1, 6.2 and 7 were written exclusively by me. Also the underlying technical work – design, implementation and running of the experiments – was conducted exclusively by me.

Section 6.3 is based on parts of the paper "On the applicability of compressive sampling in fine grained processor performance monitoring" which was submitted to ACM SIGMETRICS/Performance 2009. The paper was written jointly by me, Sean Rooney and Paul Hurley. Chapter 8 is based on the same paper.

Chapter 7 was included in the U.S. patent application Nr. CH9-2008-022, *Method and apparatus for efficient gathering of information in a multicore system*, filed in June 2008 by the IBM Corporation.

As the theoretical background of the work is very new, it is explained in a greater detail and a number of citations from the contemporary literature is given. Excerpts from the literature, such as definitions, theorems and algorithms, are carefully annotated with the original source. This allows to distinguish the contributions of this work and also helps the reader to quickly get to the seminal papers.

Chapter 2

Related work

In this chapter, we provide an overview of important areas connected to the problem of efficient gathering and delivering of performance data. In particular, we are interested in potential consumers of performance data, such as schedulers and performance monitors, and with infrastructures that have been designed to deliver or otherwise process performance data. A detailed overview of compressive sampling is given in Chapter 3.

2.1 Modern processor architectures

To effectively deliver performance data, it is crucial to understand the internal organization of the processor. In this section, we review the common aspects of contemporary processor architectures. We concentrate mainly on the the processor structure and on the parts which can significantly affect the performance.

2.1.1 Multiprocessing hierarchy

Current processors and computer systems come with a high degree of parallelism on different levels of implementation, making it natural to view the processor as a hierarchy of interconnected processing units. For instance, Figure 2.1 depicts the hardware topology of the IBM® POWER6™ microprocessor [4] and emphasizes its hierarchical organization.

Using the POWER6 as a model, we can clearly distinguish a common hierarchy of parallelization levels in the contemporary processors. At the most coarse level, we look at the microprocessor as a self-contained part of a symmetric multiprocessing (SMP) system built up from multiple processors. The processors in an SMP system share some key resources such as memory and I/O devices, but each processor executes its workload independently of the others. Going further, we realize that the processor chips are typically not monolithic. Most modern processors make use of a *chip multiprocessing* (CMP) design that integrates more execution units (execution *cores*) in one chip. These processors are often called *multicore* systems. Analogically to SMP, each core is independent in workload execution, but shares some of the on-chip resources with the other cores. Typically, a set of on-die cores shares a common L2 cache memory (see below). Independent code execution means that each core has a full set of execution units and register banks.

Some of the per-core resources can further be shared in order to achieve higher throughput and exploit the advantages of *out-of-order* instruction processing. The technique is known as *simultaneous multithreading* (SMT). In essence, it introduces a support of multiple hardware execution threads in one processor core. Each hardware thread executes given instruction stream, however, threads running in one core share common resources such as execution units and the first level instruction and data caches. This makes it possible to maximize the computational throughput by instruction reordering, so that the per-core execution units stay maximally utilized. For instance, an instruction waiting for data retrieval should not block arithmetical units for instructions which already have their data prepared. This approach can be combined with speculative execution of instructions and branch predicting mechanisms.

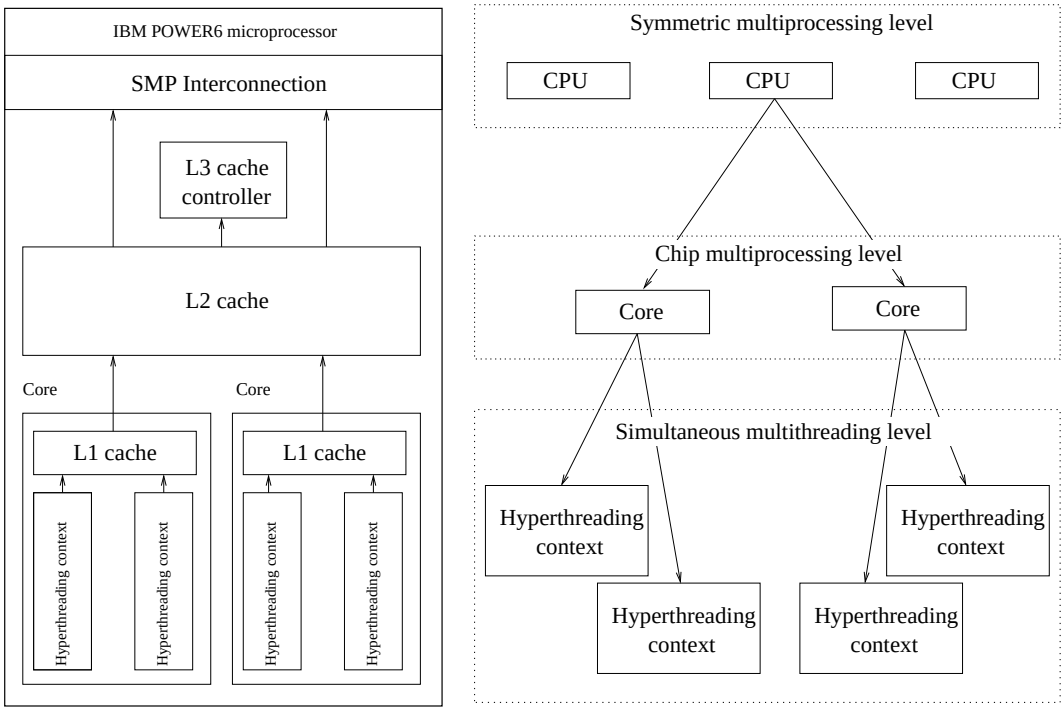


Figure 2.1: Internal structure of the IBM® POWER6™ microprocessor [4]. The picture on the left shows the logical organization of the chip, including the symmetric multi-processing (SMP) interconnection interface, L3 cache interface, two execution cores with a common L2 cache and the internal structure of the cores consisting of a dedicated L1 cache unit and two hyperthreading execution contexts. The right side of the picture shows the corresponding hierarchy of the microprocessor units.

The hierarchy of processing units can further be extended and generalized. For example, consider NUMA (non-uniform memory access) architectures [5] or wide-area architectures for high performance computing such as Ganglia [6].

2.1.2 Memory hierarchy

It is important to note that the hierarchy of processing units often combines with other hierarchical structures which are present also in flat execution topologies. A *memory hierarchy* is typically introduced to compensate for differences between the processor and memory speed. Figure 2.2 depicts a typical 2-level cached system consisting of the main memory, second level (L2) cache, first level (L1) cache and translation look-aside buffer (TLB).

We now briefly describe how a two level cache system works; the principles can easily be adapted to other cache hierarchies. In a typical situation, both the data and instructions of the program are stored in the main memory. The processor fetches the instructions from the memory and executes them. During the execution, the instructions raise requests for the data that has to be fetched from the main memory. Technical limitations cause that accessing the main memory is rather costly, as the memory can only provide a fairly slower access than is the operating frequency of the processor. Thus, it is convenient to store the instructions and data that have once been fetched in faster memories which are situated closer to the processor.

In a two level cache system, the smallest and fastest cache memory is denoted as the *L1 cache* and has typically specialized components for the instructions and data. If a processor needs to fetch something from the main memory, it first looks for it in the L1 cache. If it is found there, we say that it is a *cache hit* and the processor uses the data from the cache. Otherwise we say that there is a *cache miss* in the L1 cache and the request is forwarded to the *L2 cache* where the same procedure applies. In case of a

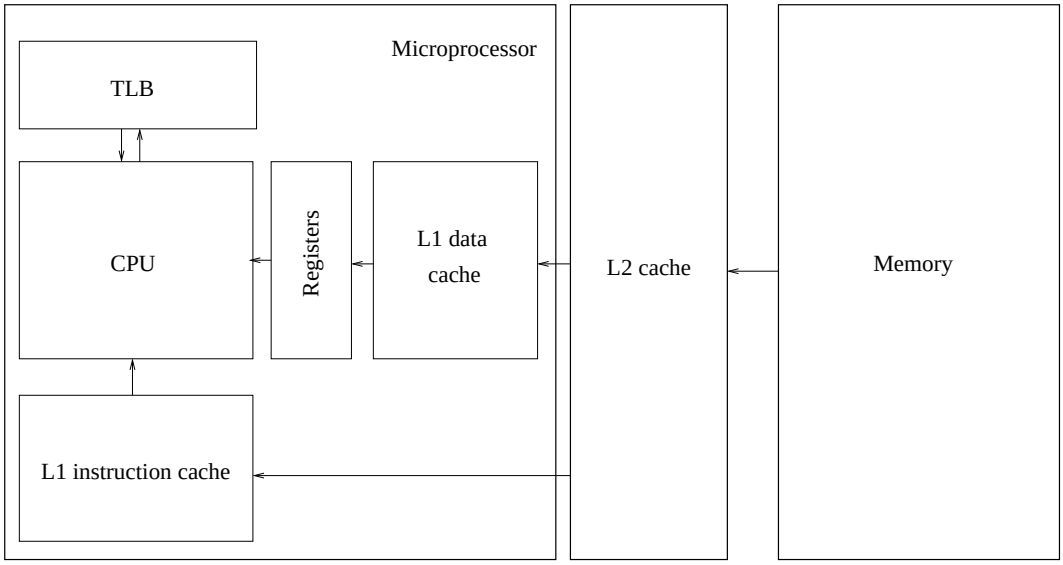


Figure 2.2: Schema of a generic 2-level cached memory system [7]. The L1 cache situated directly on the chip is divided into an instruction cache unit and data cache unit. There is also a TLB unit on the chip which facilitates fast address translation for the virtual memory subsystem. An external L2 cache is situated between the processor and the main memory.

L2 cache miss, the data is fetched from the main memory.

The translation look-aside buffer (TLB) is used when the processor supports some kind of virtual memory. Then it is often needed to translate the virtual addresses of the instructions and data to physical memory addresses. This can be done in a variety of ways. In general, the address translations cost processor time and involve memory accesses. Therefore, it is advantageous to store them for the case of reuse, which is likely to occur. Similarly to cache memories, we distinguish *TLB hits* and *TLB misses*.

In the previous description, we did not mention the problem of cache coherency, i.e. the problem of keeping all caches mutually synchronized and synchronized with the main memory. We refer the interested reader to [7] for an introductory text.

2.1.3 Performance characteristics

Having identified the key components of a processor architecture, it is natural to ask how the components affect the overall performance and what metrics can be gathered to characterize their behaviour. In the following paragraphs, we summarize the most common metrics that are useful to observe on the majority of modern processor architectures.

Overall performance characteristics

There are several quantities that are directly related to the processor performance as seen from the "outside". Most notably it is the *number of completed instructions*, i.e. the count of instructions that successfully passed all stages of the processor pipeline, all speculative phases and were really executed. One can be also interested in the *number of processor cycles* as a metric of relative time in the processor. Typically, processors can dispatch more instructions in one cycle or vice versa, one instruction can take more cycles to execute. Therefore, we often see the instructions-per-cycle (IPC) or cycles-per-instruction (CPI) ratios as the metric of overall processor performance.

Memory hierarchy characteristics

The efficiency of memory operations greatly affects the processor performance. As we explained in Section 2.1.2, the basic quantity characterizing the behaviour of a particular cache component is the number of *cache misses* or *cache hits*, respectively, number of *TLB misses* or *TLB hits*. Typically, L1 cache misses can partially be compensated for due to the instruction-level parallelism implemented in the core. Indeed, while the missing data is being fetched from the L2 cache, other instructions can possibly proceed in execution. L2 cache misses on a two level cache system cause high-latency accesses to the main memory. Operating systems often try to avoid the cache misses by implementing cache-affinity scheduling policies [8].

Pipeline stalls and branch mispredictions

The processor pipeline chains the system resources to maximize their utilization. As the stream of processed instructions is not homogeneous, smooth execution can be blocked by *pipeline stalls*. For instance, the stalls can be caused by contention for shared resources or by cache misses. Depending on the processor type, it is possible to observe the frequency of pipeline stalls, type and extent of the stalls or stall causes.

Branch prediction is a speculative mechanism designed to increase throughput of code which contains branches. Because the branches are often in an iterative code block, it is reasonable to try to predict their repetitive execution and result. *Branch mispredictions* signalize how successfully the prediction mechanism runs. A high rate of branch mispredictions causes pipeline stalls and performance drop down.

System bus characteristics

The system bus connects the processor to the memory and I/O devices. The overall count of *bus transactions* signalizes the data intensity of the workload. The ratio of *memory transactions* and *I/O transactions* helps to distinguish different types of the bus traffic, the count of *read cycles* and *write cycles* can distinguish direction of the bus traffic. This information can be used to optimize the performance, e.g. in bus traffic-aware scheduling policies [9].

Special characteristics

Some processor architectures provide special metrics reflecting various hardware or software events that might be of interest. For instance, Intel® Core 2 [10] detects a self-modifying code, which typically causes big performance penalties. Another example is counting the number of "thermal trips" which occur when a temperature limit is exceeded and the processor is forced to reduce the frequency and voltage.

2.2 Performance monitoring

The problem of monitoring processor performance, and more interestingly, monitoring the performance of internal parts of the processor infrastructure, essentially depends on the particular hardware platform. However, the basic concepts of the performance monitoring interface are similar in all current well known processors. Therefore, software abstractions emerge which make it possible to use the performance monitoring counters in more general software components, such as operating system schedulers and workload adaptation mechanisms.

2.2.1 Hardware support of performance monitoring

The Intel® Itanium® 2 processor [11] defines a performance monitoring interface which consists of two parallel sets of registers, one set for configuring the monitoring capabilities and the other to acquire the data. The monitoring concept works with the notion of *performance monitoring event*, which is also common to the majority of current processor families. Events correspond to occurrences of different internal processor activities and are counted in event counters. For instance, the corresponding event

counter is incremented whenever a cache miss in the second level cache occurs. In addition, Itanium® 2 defines *derived events* which typically provide pre-computed ratios based on the regular events, e.g. an instruction-per-cycle ratio (IPC). Itanium® 2 comes with approximately a hundred of predefined events and four 48-bit general monitoring registers. The performance monitoring events reflect instruction execution and data flow, pipeline stalls, branch predictions, data and instruction caching, system bus traffic and other. Facilities to perform filtering or additional information recording are present.

The IBM Cell Broadband Engine™ processor [12] defines several hundreds of performance monitoring events that can be recorded to eight 16-bit counters, resp. to four 32-bit counters. Similarly to Intel® processors, the events cover the most influential processor components, including the execution pipelines, memory architecture, buses, I/O controllers and other. The counters can be configured through a set of control registers. Counter values can be collected in defined intervals and transferred to a *trace array*, which can store up to 1024 128-bit values.

The AMD Opteron processor [13] provides a simple performance monitoring interface with four general 48-bit event counters controlled by a parallel set of configuration registers. The events span all important processor components, including the execution pipeline, memory hierarchy and I/O system. The Opteron processor provides only a minimal additional functionality above the basic monitoring.

The MIPS R10000 processor [14] implements a minimalistic performance monitoring unit consisting of two 32-bit counters, two control registers and approximately 30 monitorable events.

2.2.2 Software support of performance monitoring

Despite the tight connection of performance monitoring to the processor microarchitecture, there have been continuous efforts to provide a software abstraction over different processor families and models. In this section, we mention software layers which were designed to directly access performance counters, but without the need to write machine dependent code.

On the driver level, there are two widely used open source projects for the Linux® operating system. Stephane Eranian's *perfmon* [15] currently supports all modern Intel® processors (including the Itanium®, Core 2 and P4 processor families), AMD Opterons and AMD K7 processors, IBM® POWER, Cell Broadband Engine™ and PPC processors and MIPS processors. The interface generalizes some of the common concepts in the processor families and exports logical registers for the monitoring control and data acquisition. The core interface defines methods for manipulating the monitoring context, data and control registers and events. The monitoring context can be local (per-thread) or system-wide.

Mikael Pettersson's *perfctr* [16] supports Intel® x86 processor family (including P4 and Core 2 processor families, but not including Itanium® processors), AMD K7 and K8 processors, Cyrix and IBM PPC processors. Similarly to *perfmon*, the interface provides logical control and data registers which are mapped to the machine-dependent monitoring unit.

The PAPI project [17] aims to define a standard API for accessing performance monitoring capabilities. The PAPI architecture defines a portable two-layered application interface and a machine dependent substrate. Currently, the substrates cover Intel®, AMD, IBM, MIPS, Cray and UltraSparc processors and are ported to Linux®, Windows, Solaris, AIX and other operating systems. When compared to [15, 16], the PAPI interface provides a higher level cross-platform support. It allows the user to work with event sets with only a little knowledge of the underlying hardware constraints; on the other hand, it hides rich platform specific features that can be more easily exported by a low level driver. For instance, *perfmon* for Intel® Itanium® 2 [11] exports the EAR registers which allow the user to record the instruction addresses which caused the cache or TLB misses.

2.3 Operating system schedulers

Operating system schedulers can be considered a natural consumer of the real-time performance information. We find it useful to look at the world of schedulers from the following perspectives:

(i) Hardware topology

(ii) Scheduling objectives

(iii) Historical development

Hardware topology. In Section 2.1, a hierarchical view of the processor internal structure was introduced. In particular, we distinguished the *symmetric multiprocessing* (SMP) level, *chip multiprocessing* (CMP) level and *simultaneous multithreading* (SMT) level. Depending on the thoroughness of the scheduling algorithm, different inputs may be considered and different strategies may be implemented to take such a structured hardware topology into account. For instance, CMP-aware schedulers can take advantage of the shared L2 cache which allows for an efficient communication among the tasks allocated to the cores in one processor package. SMT-aware schedulers can apply policies involving frequent task switches, because a task switch within an SMT domain is cheap. In contrast, schedulers for SMP machines with non-uniform memory access (NUMA) must consider varying costs involved in the task switch.

Scheduling objectives are in general disjunctive and schedulers have to decide on their objectives according to the application domain they are targeting. For instance, achieving a high throughput is valuable in high performance computing but may be considered less important in interactive applications that require good responsiveness. Once the semantic objective is selected, the technical criterion to achieve the objective has to be considered. Typically, schedulers tend to gain a response from the system to heuristically adjust their decisions. For instance, the way of using the assigned time quanta signalizes the interactivity degree of the application. In the past years, the microarchitectural performance monitoring has opened new possibilities for the schedulers. It is now possible to monitor behaviour of the caches, system bus and internal processor structures and dynamically adapt scheduling according to that.

Historical development of scheduling algorithms is also bound to the development of hardware. In what follows, we would like to emphasize the transition from the traditional "oblivious" [18] schedulers to the schedulers which respect the hardware structure and adapt to the workload. This development is particularly forced by the increasing complexity of hardware architecture and is facilitated by the new performance monitoring capabilities. It should be pointed out that due to the extensive and specific requirements on real operating system schedulers, only a few ideas with the best enhancement/cost ratio have been implemented as a real world scheduling policy.

2.3.1 Traditional schedulers

Traditionally, the task scheduling problem [5] considered a scenario of a multiprogrammed workload competing for a single processor. Various scheduling algorithms were developed for system classes with differing requirements, such as the batch systems, interactive systems and real-time systems. The traditional algorithms try to heuristically solve a set of disjunctive goals, including maximization of throughput, minimization of response time, keeping fairness and assuring proportionality defined by the user. Many of the very basic algorithms (see [5]) form the foundation of today's real world operating system schedulers (e.g. [19]). From our point of view, it is important to observe that the classical single-CPU scheduling algorithms are very loosely bound to the underlying hardware; typically, their input consists of the task lists, task prioritization and feedback gained from observing how the assigned time quanta are consumed by the tasks.

2.3.2 Symmetric multiprocessing schedulers

The advent of shared-memory multiprocessor configurations (SMP) advanced the scheduling complexity, forcing the scheduler to map a set of tasks to a set of processors. According to the topology of the system, multiple costs and consequences of the task switches have to be considered. Tucker and Gupta [20] observe a significant degradation of parallel application performance when the underlying scheduling algorithm implements only a simple round-robin algorithm and multiple processes contend for multiple processors. This fact triggered the development of schedulers that are more aware of the underlying hardware topology. Squillante and Lazowska [8] propose to avoid the corruption of per-processor caches

by using processor cache affinity strategies in schedulers. This idea is also present in the current real world schedulers (e.g. [19]).

Cache-affinity scheduling [8] has since become a standard way of task scheduling on SMP architectures [19] and also many proposed scheduling algorithms target the cache behaviour as one of the most important performance predictors (e.g. [21, 22, 18, 23]). But the cache performance is apparently not the only usable predictor. For instance, Antonopoulos et al. [9, 24] suggest to consider processor-memory bus performance as a potential bottleneck of the SMP and thus, as a controlling variable for the scheduling mechanisms. They proposed scheduling algorithms which use hardware performance counters to estimate the bus bandwidth of applications.

2.3.3 First performance-aware schedulers

The obvious fact that scheduling performance is influenced mainly by the concrete workload brings the attention of many researchers to the problem of adapting the scheduling mechanisms to a particular application requirements. Corbalan et al. [25] published scheduling mechanisms controlled by dynamically observed application characteristics. They based their scheduling policies on the speedup of parallel program regions and several other criterions reported to the scheduler by runtime libraries included in the scheduled programs. This pattern appeared in many later works which, more interestingly, considered hardware provided application characteristics instead of artificially computed ratios.

2.3.4 Simultaneous multithreading schedulers

Increased parallelism at subtle hardware levels introduced by multithreading techniques (such as simultaneous multithreading (SMT), [26]) triggered the emergence of the work on *symbiotic jobscheduling* [23, 27]. The main idea is to search for a symbiotic schedule, i.e. a schedule consisting of properly selected threads which leads to low processor resource contention and higher throughput. The proposed scheduling algorithm consists of three phases called sampling, optimization and symbiosis. In the sampling phase, the schedules are randomly perturbated and the performance characteristics from hardware counters are collected, such as the number of instructions per cycle, number of cache hits, number of conflicts in the floating point units etc. In the optimization phase, the schedule that is believed to be the most symbiotic one is picked up and run in the symbiosis phase. Only single processor systems are considered.

Parekh et al. [18] proposed scheduling policies for SMTs designed to be driven by per-thread performance metrics. The metrics are sampled from hardware performance counters and include the miss rates of the L1 and L2 caches, instructions per cycle ratio, memory access time and other. The proposed algorithms utilize greedy design patterns, which means the typically schedule the threads with the most optimal values of the observed ratios. Only single processor systems are considered.

2.3.5 Schedulers for symmetric multiprocessing with SMT processors

Nakajima et al. [22] target the multiprocessor multithreaded environment and propose a load-balancing scheduling assistant. The assistant relies on hardware-provided performance metrics including the cache misses, number of load and store operations, bus activity and other. The processors with high probability of resource contention are found by defining a threshold for each of the performance metrics. Once a resource contention is detected, the processors with the highest and the lowest load are found (processor load is computed from the hardware performance metrics) and a pair of processes is swapped between the two most unbalanced processors.

McGregor et al. [28] proposed a scheduling policy based on the concept of thread pairing, where threads with opposite extremal values of selected performance metrics are paired and scheduled together on one single processor. The suggested performance metrics are the rate of stall cycles, rate of cache misses and rate of bus transactions. The rates are based on the hardware performance monitoring. To avoid noise in the observations, performance metrics are gathered with a small history window and a smoothing function (moving average) is applied.

2.3.6 Chip multi processing schedulers

Recent processor architectures which involve multiple SMT processors in a chip multiprocessing environment are considered by Fedorova et al. [29]. They propose a two-phase scheduling algorithm which aims at fair usage of the second level caches, which are shared among the multiple cores in one processor package. The algorithm uses a simple analytical cache model based on the hardware performance counters.

DeVuyst et al. [30] extend the algorithms of symbiotic scheduling [23, 27] and suggest improving the search space of possible schedules by considering the unbalanced schedules as well. Their scheduler estimates processor performance and power requirements by using hardware counters.

Results on the design of multi-core architectures with a heterogeneous core set are published by Kumar et al. [31]. A scheduling policy is proposed which uses hardware performance counters in sampling phase of a two-phase scheduling algorithm to estimate execution profile of the workload.

2.4 Power-aware computing

As Chapter 1 mentions, the power consumption of modern processors is often the limiting parameter of their performance. Subsequently, it is desirable to perform a careful analysis of the power consumption and temporal and logical distribution of the energy in the processor. Moreover, power consumption of a running system can be effectively influenced by software.

Joseph and Martonosi [32] discuss the problem of conversion between the commonly available hardware performance counters and metrics of dissipated power for a particular processor component. They construct a mapping between the processor performance counters and per-unit power metrics. Similar analytical power models are concerned in Kadayif et al. [33]. The approach is later revisited for the case of more recent processors with better performance counters by Isci and Martonosi [34], who provide per-component power weightings for Intel® Pentium 4 processors and use them to compute a detailed decomposition of their per-processor power measurements (which were obtained by using a physical measurement procedure). In all these works, the hardware performance counters are involved as the essential source of insight into the internal structures of the processor.

Curtis-Maury et al. [35] propose algorithms for self-adaptation of parallel programs. They target SMP machines built up from SMT processors. They use information from the hardware performance counters to predict performance of parallel regions and balance the performance and power consumption. Apart from the frequently used instruction-per-cycle ratio, more advanced metrics are concerned (e.g. the rate of mispredicted branches).

Weissel and Bellose [36] suggest to continuously adapt the processor frequency to the observed run-time behaviour of the workload. The processor frequency is scaled on each task switch with the goal of optimizing the power consumption and preserving the performance degradation in given boundaries. The run-time thread execution profiles are obtained by sampling the hardware performance counters.

Power density and overheating problems of multicore processors with SMT cores are targeted by Powell et. al. [37]. Two complementary thread assignment strategies are proposed, one that maximizes the usage of the per-core resources so that the overheated units cool down simultaneously, the second that migrates the threads of overheated cores to more suitable cores. In both strategies, the hardware performance counters provided by the processor are used to discover the run-time characteristics of the threads and cores (e.g. instructions-per-cycle ratio, integer/floating-point character, resource usage).

2.5 Wide-area performance monitoring

When seen from a greater distance, the problems and logical structures of wide-area performance monitoring are analogical to those of chip multiprocessing. In both domains, the task is to deliver the performance data from multiple nodes interconnected by a shared data channel while minimizing the impact on the performance and consumed bandwidth.

Mooney et al. [38] present a cluster monitoring framework designed with the goal of supporting fine grained performance data delivery. The framework consists of a set of client modules and of a

data collection server. The client modules sample the CPU performance data and the operating system status which is then encoded into an XDR representation and multicasted by the UDP protocol to the server. Performance metrics include the bytes-per-cycle ratio for memory access and the number of *flops* (floating-point operations per second). The hardware performance counters are used as a source for the performance metrics.

Ganglia monitoring framework [6] is a distributed monitoring system with an emphasis on scalability and hierarchical organization. In Ganglia, the local node information is collected by a monitoring daemon and communicated by multicast packets within the originating cluster. A set of clusters can be interconnected to a federation using a *meta daemon* which polls information from the representative cluster nodes. The data is encoded in the XDR and transferred in XML representation. The built-in performance metrics capture mainly the operating system status (memory, processor utilization, clock) and CPU load; they can be extended by a set of user-defined metrics.

Similar communication patterns can be found in other cluster monitoring systems, such as [39, 40, 41].

Chapter 3

Compressive sampling

Many scientific and engineering areas deal with the problem of signal processing. Real world signals often need to be captured, transferred and processed. As an example, consider sound recordings, photography, medical imaging, navigation etc. Because digital resources are very limited in their resolution and bandwidth, signals are sampled and often compressed.

Traditionally, signal sampling and compression phases are treated separately. First, the signal is sampled and consecutively, the compression is applied. In many cases, this involves unnecessary collection of the whole signal, which is then compressed and its substantial part is thrown away. Compressive sampling [1, 2, 3] tries to answer the question, whether it is possible to compress a signal directly in the sampling phase in such a way that only the data that suffice to reconstruct the signal is transferred. Under certain assumptions, it is possible to achieve this with a high probability.

3.1 Basic concepts

In the following text, we concentrate on the task of sampling, compression and recovery of a discrete signal $f \in \mathbb{R}^n$.

By correlating the signal to a set of basis vectors $\{\psi_k\}$, $k = 1, ..., m$, $\psi_k \in \mathbb{R}^n$, we obtain samples $y \in \mathbb{R}^m$:

$$y_k = \langle f, \psi_k \rangle, k = 1, ...m$$

We will denote this procedure as *signal sampling*.

Note that not necessarily $m = n$; we will be interested in cases when $m << n$. This setup is sometimes described as signal undersampling and in compressive sampling, it is the way to achieve signal compression.

Signal recovery takes a vector of samples $y \in \mathbb{R}^m$ and reconstructs the signal estimate f', such that the constraint $y_k = \langle f', \psi_k \rangle, k = 1, ...m$ holds. As long as $m < n$, there are infinitely many candidates for f'.

3.2 Sparsity of the input signal

Compression principles are often based on the fact that the signal being compressed has a concise representation in some basis (sometimes called domain). For instance, natural images are known to be concisely representable in the wavelet domain [42]. In order to develop a compression theory and to provide theoretical guarantees of the signal recovery, we have to somehow formally express the notion of *compressibility*. The formal model of a compressible signal will be then a part of the assumptions in the theoretical guarantees of recovery.

Let us distinguish the following definitions.

(i) Sparse signal

(ii) Approximately sparse signal

(iii) Compressible signal

3.2.1 Sparse signal

A sparse signal has a concise representation in a given basis. Technically speaking, this means that when the signal is expressed as a vector in $\mathbb{R}^n$, it uses only a limited portion of the coefficients. We will use the following sparsity definition, so that we can state how much a signal is sparse [1]:

Definition 1. Let $f \in \mathbb{R}^n$ be a vector and Ψ an orthonormal basis consisting of basis vectors $\psi_1, ..., \psi_n$. Let $x \in \mathbb{R}^n$ be a representation of vector f in basis Ψ, i.e.

$$f = \sum x_i \psi_i$$

We say that f is S-sparse in Ψ if $|\{j; x_j \neq 0\}| \leq S$.

Informally speaking, an S-sparse signal has at most S non-zero entries in its coefficient vector. Note that the notion of sparsity is tightly bound to the chosen basis. Finding the basis providing a concise (sparse) representation of the signal is important for the compression to work. Refer to Section 3.3 to see the role of sparsity in compressive sampling assumptions.

3.2.2 Approximately sparse signal

An approximately sparse signal does not obey exactly the strict definition of sparsity. This model is practical when we work with real world signals. Assume we have a signal f and a sparsity level S. Then, let us denote f_S the signal which we obtain if we take only the S largest coefficients of f in some basis Ψ. If the signal f is S-sparse in Ψ, we have clearly $\|f - f_S\| = 0$. However, if the signal is not exactly S-sparse, we can use the l_p norm $\|f - f_S\|_{l_p}$ as an indicator of the noise introduced when we assume the signal is S-sparse. If we can find reasonable bounds for $\|f - f_S\|_{l_p}$, we say that f is *approximately sparse*.

We will see that in many cases, the theoretical guarantees on the recovery stability are provided with respect to the deviation of the signal from the assumed sparsity. For instance, we can say that the recovery is stable if the reconstructed signal differs from the original one only proportionally to $\|f - f_S\|_{l_1}$. See Theorem 2 for a stability result on the l_1-minimization recovery.

3.2.3 Compressible signal

The notion of signal compressibility offers the biggest interpretation freedom of the three definitions we mention. Sparse signals are compressible by definition, because we can represent them only by their non-zero coefficients. However, a compressible signal does not have to be necessarily sparse; instead, it can be sufficient when its coefficients decay steeply. We will restrict ourselves to the definition provided by Candès in [43]:

Definition 2 (Signal representation in a weak ball of given radius, [43])**.** Let $f \in \mathbb{R}^n$ be a vector and Ψ an orthonormal basis consisting of basis vectors $\psi_1, ..., \psi_n$. Let $\Psi(f)$ be representation of f in Ψ. Order the coefficients of $\Psi(f)$ such that

$$|\Psi(f)|_{(1)} \geq |\Psi(f)|_{(2)} \geq ... \geq |\Psi(f)|_{(n)}$$

We say that $\Psi(f)$ belongs to the weak l_p ball of radius R, if

$$|\Psi(f)|_{(i)} \leq R \cdot \frac{1}{i^{1/p}} \quad \text{for all} \quad 1 \leq i \leq n$$

where $p > 0$, $R > 0$.

A signal belonging to a weak l_p ball has coefficients in Ψ that *decay like a power law*. The p parameter controls the speed of the decay. Candès proves the immediate consequences of this property [43], we have

$$\|f - f_S\|_{l_2} \leq C \cdot R \cdot S^{-p}$$

We see that this type of compressible signal is approximately sparse.

3.3 Signal reconstruction theorems

In this section, we summarize the theoretical background of compressive sampling. The theorems are categorized according to the characterization of the sampling matrix. First, the RIP property is introduced which can be used as a characterization of any sampling matrix. Subsequently, the theorems for more specific sampling matrices are listed, including the orthogonal, random and incoherent sampling matrices.

3.3.1 General sampling matrices

Consider the task of recovering a signal f with a sparse representation in basis Ψ from the measurements $y = \Psi f$. We will see that it is possible to recover the signal exactly if the signal is sparse and the sampling matrix Ψ meets certain assumptions.

Definition 3 (Restricted Isometry Property, [44])**.** Let $\Psi = \{\psi_j\}_{j \in J}$ be a matrix with $|J|$ columns. For every integer $1 \leq S \leq |J|$, the number δ_S is the smallest quantity such that

$$(1 - \delta_S)\|c\|^2 \leq \|\Psi_T c\|^2 \leq (1 + \delta_S)\|c\|^2$$

for all column subsets $T \subset J$, $|T| \leq S$, and all real coefficient vectors $(c_j)_{j \in T}$. We say that δ_S is the S-restricted isometry constant of Ψ.

Given the matrix Ψ and the column count S, small values of δ_S indicate that every column set with at most S columns drawn from Ψ behaves nearly like an orthonormal system. Indeed, multiplication by an orthonormal matrix does not change vector length [1] and this property is approached the more the δ_S is numerically lower.

Theorem 1 (Exact recovery of an exactly sparse signal, [44])**.** *Let Ψ be an n-column matrix, $S \geq 1$, $f \in R^n$ an S-sparse vector, $y = \Psi f$ the measurement vector. If $\delta_{2S} + \delta_{3S} < 1$, then the solution f' to the l_1 minimization recovery task*

$$\min_{f' \in \mathbb{R}^n} \|f'\|_{l_1} \quad \textit{subject to} \quad \Psi f' = y$$

is exact.

The theorem says that if the given matrix has sufficiently small S-restricted isometry constants (which in language of [43] means that the matrix obeys the uniform uncertainty principle), then the l_1-norm optimization yields an exact reconstruction of all S-sparse signals. Note that for practical usability, we have to find a sampling matrix fulfilling given assumptions and we can work only with signals that are exactly S-sparse. However, this is often not the case, as many real world signals tend to only be approximately S-sparse.

Having a vector f, we will denote f_S the vector obtained by setting all but the biggest S coefficients of f to zero.

Theorem 2 (Stable recovery of an approximately sparse signal, [45])**.** *Let Ψ, S, f and y be set up as in the previous theorem. If $\delta_{3S} + \delta_{4S} < 2$, then the solution f' to the l_1 minimization recovery task obeys*

$$\|f' - f\|_{l2} \leq C \cdot \frac{\|f - f_S\|_{l1}}{\sqrt{S}}$$

The expression $\|f - f_S\|_{l1} = \sum_i |(f - f_S)_i|$ reflects the amount of noise introduced by the "sparsification" of f. If f is already S-sparse, the theorem says $\|f' - f\|_{l2} = 0$, which is a statement about an exact reconstruction of an exactly S-sparse signal which we obtained earlier. However, when f is not exactly S-sparse, we get an upper bound on the reconstruction error which is proportional to how much the signal f differs from assumed sparsity. The constant C behaves reasonably [45]. The stability can also be expressed using the l_1 norm [2].

[1] By definition, orthogonal matrix A is a matrix such that $A^T A = I$. When applying it to vector v, the length persists: $\|Av\|_{l2} = \|v\|_{l2}$. This is because $\|Av\|_{l2}^2 = (Av)^T(Av) = (v^T A^T)(Av) = v^T(A^T A)v = v^T v = \|v\|_{l2}^2$.

3.3.2 Orthogonal sampling matrices

The restricted isometry property is a general criterion on the sampling matrix. If we want to be more concrete, we may restrict ourselves to the class of orthogonal sampling matrices. By doing that, we also get better understandable estimates on the number of samples that is required to achieve a perfect reconstruction.

Theorem 3 (Exact recovery of an exactly sparse signal, orthogonal sampling matrix, [46])**.** *Let Ψ be an orthogonal matrix of size $n \times n$ with entries of magnitude $O(1/\sqrt{n})$. Let Ω be a randomly chosen subset of measurement vectors from Ψ of size m. Let f be an S-sparse real signal of length n. If*

$$m = O(S \log^4 n)$$

then f can be exactly reconstructed with high probability from measurements $y = \Psi_\Omega f$ by solving the l_1 minimization recovery task

$$\min_{f' \in \mathbb{R}^n} \|f'\|_{l_1} \quad \textit{subject to} \quad \Psi_\Omega f' = y$$

We see that the number of measurements depends linearly on the signal sparsity and polylogarithmically on the length of the signal. To illustrate how assumptions of the theorem can be varied, consider that the coefficient set T is known which forms the support of the signal in the sparsity domain. Then the assumptions are stronger: where Theorem 3 requires the conclusions to hold for all possible T's, we get a tighter bound if the T is fixed. Moreover, a broader class of sampling matrices can be considered. Suppose we measure the "flatness"' of the sampling matrix in the following way:

Definition 4 (Matrix concentration, [47])**.** Let Ψ be a matrix, we say that

$$\mu(\Psi) = \max_{ij} |\Psi_{ij}|$$

is the matrix concentration of Ψ.

If we follow the conventions from [47] and normalize Ψ such that $\Psi^* \Psi = nI$ [2] , we get the bound

$$1 \leq \mu(\Psi) \leq \sqrt{n}$$

The value of $\sqrt{n}$ is achieved if the matrix contains a row with only one non-zero entry (that must be of size $\sqrt{n}$). The value of 1 is achieved by "'flat"' matrices.

Theorem 4 (Exact recovery of an exactly sparse signal, orthogonal sampling matrix and fixed signal support, [47])**.** *Let Ψ be an orthogonal matrix of size $n \times n$ normalized such that $\Psi^* \Psi = nI$. Let T be a fixed support set of signal domain, let $\{z\}$ be a sequence of +-1 drawn from symmetric Bernoulli distribution (i.e. $P(z_i = 1) = P(z_i = -1) = \frac{1}{2}$). Let Ω be a randomly chosen subset of measurement domain of size m. If*

$$m \geq C_0 \cdot |T| \cdot \mu(\Psi)^2 \cdot \log \frac{n}{\delta}$$

and

$$m \geq C_0' \cdot \log^2 \frac{n}{\delta}$$

for some fixed C_0, C_0', then every signal f supported on T with signs matching z can be recovered from measurements $y = \Psi_\Omega f$ by solving linear l_1 minimization recovery task

$$\min_{f' \in \mathbb{R}^n} \|f'\|_{l_1}$$

subject to $\Psi f' = y$ with probability exceeding

$$1 - \delta$$

We include this theorem in exact wording from [47], so that the underlying technical means of describing the signal set are visible. The theorem says that in order to have the l_1 reconstruction working with high probability, we have to choose the number of measurements depending on the "quality" of the sampling matrix, required probability and signal sparsity.

[2] Ψ^* is the conjugate transpose of Ψ

3.3.3 Random sampling matrices

It was shown that for several types of random matrices, the assumptions of the previous theorems hold with a high probability [43]. This is interesting as the matrices are completely unstructured, produce noise-like, universal measurements and can be effectively implemented in hardware (see Chapter 7).

Theorem 5 (Compressive sampling of exactly sparse signals with random matrices, [43]). *Let Ψ be a $K \times N$ matrix with elements sampled independently and identically from*

1. *the normal distribution $N(0, \frac{1}{K})$ or*
2. *the symmetric Bernoulli distribution, i.e. $P(\Psi_{ij} = \frac{1}{\sqrt{K}}) = P(\Psi_{ij} = -\frac{1}{\sqrt{K}}) = \frac{1}{2}$*

Then for a sufficiently sparse signal, i.e. an S-sparse signal with

$$S \leq C \cdot \frac{K}{\log \frac{N}{K}}$$

the compressive sampling recovery Theorems 1, 2 hold with probability

$$1 - O(\exp^{-\gamma N})$$

for some $\gamma > 0$.

The following theorem summarizes the stability of l_1 recovery for *compressible* signals, i.e. for signals that are not sparse, but belong to a weak l_p ball (see Definition 2). Thus, we assume that the signals have rapidly decaying coefficients in some basis; this is a somewhat weaker assumption than assuming a sparsity.

Theorem 6 (Compressive sampling of compressible signals with random matrices, [43]). *Let Ψ be a $K \times N$ random matrix and let $f \in R^n$ be in a weak l_p ball of some radius R with decay speed $0 < p < 1$. Choose a precision constant $\alpha > 0$. With probability at least $1 - O(n^{-\rho/\alpha})$, the solution f' to the l_1-minimization task can be bounded as*

$$\|f - f'\|_{l_2} \leq C \cdot R \cdot (\frac{K}{\log N})^{-r}$$

where $r = \frac{1}{p} - \frac{1}{2}$. The constant C depends on the decay speed p and precision constant α.

3.3.4 Incoherent sampling and recovery matrices

In the previous sections, we have considered the setup of recovering a signal f with a sparse representation in a basis Ψ from the measurements $y = \Psi f$. But it may happen that the signal f is not sparse in the measurement basis Ψ, however, we know that it is sparse in some other basis Φ. We can use the same sampling process involving only the matrix Ψ and change the recovery process to use both the bases Ψ and Φ. We then obtain the signal represented in Φ - so from now on, we will call Φ the *representation basis*. The important result is that when the pair of sampling matrix and representation matrix obeys certain properties, we can use the compressive sampling theorems stated above.

We will now define a measure of basis coherency, to be able to express constraints on mutual representability of a pair of bases.

Definition 5 (Mutual coherence, [1]). Let $\Psi = \begin{bmatrix}\psi_1 & \psi_2 & ... & \psi_n\end{bmatrix}$, $\Phi = \begin{bmatrix}\phi_1 & \phi_2 & ... & \phi_n\end{bmatrix}$ be orthonormal bases of $\mathbb{R}^n$. The mutual coherence of Ψ and Φ is

$$C(\Psi, \Phi) = \sqrt{n} \max_{1 \leq k,j \leq n} |\langle \psi_k, \phi_j \rangle|$$

Note that the absolute value of the scalar product reflects the linear dependency of the two vectors. By taking into account the maximum dependency between vectors of given bases, mutual coherency expresses how much a basis can be expressed using the other one. Mutual coherence has higher values for bases with correlated vectors and lower values for the other ones. It is then useful to know that

Theorem 7. *For any Ψ, Φ bases of $\mathbb{R}^n$, $1 \leq C(\Psi, \Phi) \leq \sqrt{n}$*

Now an analogy of the Theorem 4 can be derived for a structured recovery matrix. Suppose we sample randomly in the $\Psi = \begin{bmatrix}\psi_1 & \psi_2 & ... & \psi_n\end{bmatrix}$ domain:

$$y_j = \langle f, \psi_j \rangle \text{ for } j \in \mathcal{M}$$

where $\mathcal{M}$ selects a subset of $m = |\mathcal{M}|$ measurements. The recovery matrix is changed to $\Psi\Phi$ and the recovery task reformulated as

$$\min_{f' \in \mathbb{R}^n} \|f'\|_{l_1} \quad \text{subject to} \quad \langle \psi_j, \Phi f' \rangle = y_j \text{ for all } j \in \mathcal{M}$$

Note the form of the minimization constraint. Its purpose is to keep the signal estimate f' coherent with the measurements y. First, the signal estimate f' is transformed to the Φ domain and consequently, it is put through the sampling process and correlated to the measurements. Let us conclude this to the following theorem:

Theorem 8 (Compressive sampling with incoherent bases, [47])**.** *Let $\Psi = \begin{bmatrix}\psi_1 & \psi_2 & ... & \psi_n\end{bmatrix}$, Φ be orthonormal n-dimensional bases, $f \in \mathbb{R}^n$ be a K-sparse vector in Φ. Let $y \in \mathbb{R}^m$ be a measurement vector obtained by selecting m measurements in Ψ domain randomly and uniformly, i.e. $y_j = \langle f, \psi_j \rangle$ for $j \in \mathcal{M}$, $\mathcal{M}$ indexing the subset of columns of Ψ. If*

$$m \geq \epsilon * C(\Psi, \Phi)^2 * K * \log \frac{n}{\delta}$$

for some $\epsilon > 0$, $\delta > 0$ then the solution to l_1 minimization recovery task

$$\min_{f' \in \mathbb{R}^n} \|f'\|_{l_1} \quad \textit{subject to} \quad \langle \psi_j, \Phi f' \rangle = y_j \textit{ for all } j \in \mathcal{M}$$

is exact with probability exceeding $1 - \delta$. The result holds for a fixed support and signals with signs matching a fixed sign sequence as in the Theorem 4.

For approximately sparse signals, the results of the Theorem 2 hold analogically.

3.4 Recovery algorithms

The acquisition part of compressive sampling framework consists of correlating the signal f to given sampling matrix Ψ. The measurement vector y is then transferred to the recovery algorithm whose task is to expand it to the best possible approximation f' of the original signal f. This is a very demanding task; a straightforward solution would be of exponential time complexity. Indeed, the system of equations

$$y = \Psi f'$$

itself is ill-posed for sampling matrices Ψ with k rows and n columns, when $k < n$. However, we are naturally interested in matrices with more columns than rows, because we use them to "extract" important information from a signal of length n and to produce its compressed form of length k. In compressive sampling, we use the additional a priori assumption that the signal f is sparse and thus we are searching only for sparse solutions f'. If we know this, we can rewrite the recovery task as

$$\min \|f'\|_{l_0} \quad \text{subject to} \quad y = \Psi f'$$

where $\|f'\|_{l_0} = |\{i; f'_i \neq 0\}|$ is the l_0 quasi-norm counting the number of non-zero elements in f'. By minimizing the l_0-norm, we are searching for the sparsest candidate fulfilling given constraint. However, this is a task of combinatorial complexity that is not solvable in practical applications. Therefore, multiple approaches have been developed to deal efficiently with the recovery of sparse signals.

3.4.1 Optimization with relaxed objective or constraint function

The straightforward l_0-minimization task can be refined in two ways to make it computationally feasible. Either the objective function (the l_0 sparsity proxy) can be altered or the optimization constraint can be relaxed. We give two representative approaches, Basis Pursuit and Gradient Projection, along with their variations. The *Basis Pursuit* [48] uses the l_1-norm ($\|f'\|_{l_1} = \sum_i |x_i|$) instead of the l_0-norm as a proxy for signal sparsity. This makes it possible to recast the task as a linear program. *Gradient projection* methods recast the recovery task as an unconstrained quadratic program which is solved iteratively by projecting the gradient of the objective function onto the set of feasible solutions.

Basis pursuit with equality constraints

Basis Pursuit was first introduced in [48] as a principle of signal decomposition into atoms of given overcomplete dictionary. The decomposition is required to be optimal in the sense that its l_1 norm is minimized. The key idea is that the l_1 minimization promotes the sparse decompositions.

Basis pursuit with equality constraints is the basic setup designed for an *exact* signal recovery.

Definition 6 (Basis pursuit with equality constraints, [48])**.** Given the signal f, recovery matrix Ψ and measurement vector y, we say that $\min\|f'\|_{l_1}$ subject to $\Psi f' = f$ is the basis pursuit with equality constraints.

Theorem 1 states that it is possible to exactly recover a sufficiently sparse signal by a basis pursuit, provided that the sampling matrix Ψ fulfills given assumptions. If the signal is not exactly sparse, the recovery method is still reasonably stable, see Theorem 2. Theorem 8 summarizes conditions under which the basis pursuit can be used to recover measurements in an incoherent basis with a high probability of success.

Basis pursuit with quadratic constraints

In practice, it can be desirable to relax the equality constraints prescribed by the elementary form of basis pursuit (Definition 6). In particular, if the observed data is distorted by a deterministic or stochastic unknown error term e with known bounds, i.e. $\|e\|_{l_2} \leq \epsilon$, a *quadratic constraint* can be used.

Definition 7 (Basis pursuit with quadratic constraints, [45])**.** Given the signal f, recovery matrix Ψ, measurement vector y and error bound ϵ, we say that $\min\|f'\|_{l_1}$ subject to $\|\Phi f' - y\|_{l_2} \leq \epsilon$ is the basis pursuit with quadratic constraints.

The constraint is described as "quadratic" because it uses the l_2-norm to measure the recovery error, $\|f'\|_{l_2} = \sqrt{x_1^2 + ... + x_n^2}$. Therefore, bigger deviations are quadratically penalized, smaller deviations are neglected.

According to [45], recovery of exactly sparse signals using the basis pursuit with quadratic constraints is possible with an error that is proportional to the amount of noise ϵ. For signals that are not exactly sparse, the recovery error is proportional to the noise ϵ and noise introduced by the sparsity assumption (see Theorem 2 of [45]).

Basis pursuit with bounded residual correlation (Dantzig selector)

Candès and Tao named their estimator proposed in [49] the Dantzig selector as a tribute to the inventor of linear programming. The idea is to relax the linear programming constraint in such a way that the residual vector $y - \Psi f'$ is not much correlated to the recovery matrix Ψ.

Definition 8 (Dantzig selector, [49])**.** Given the signal f, recovery matrix Ψ, measurement vector y and error bound γ, we say that $\min\|f'\|_{l_1}$ subject to $\|\Psi(\Psi f' - y)\|_{l_\infty} \leq \gamma$ is the Dantzig selector.

The l_∞-norm is defined as

$$\|f'\|_{l_\infty} = \sup_{1 \leq i \leq p} |x_i| \leq \lambda_p \sigma$$

There are proposed several reasons for using such an estimator [49]; among others, the estimation procedure is not sensitive to any orthogonal transformations applied to the data vector. The Dantzig selector can be recast to a linear programming problem.

Candès and Tao show that for models where measurements are distorted by Gaussian noise with known variance, i.e. the measurements are of form $y = \Psi f + \epsilon$, $\epsilon \sim N(0, \sigma^2 I_n)$, the Dantzig selector recovers with high probability with error proportional to variance σ^2, sparsity and logarithm of signal length [49].

Gradient projection for sparse reconstruction (GPSR)

Let Ψ be a $k \times n$ matrix, let y be the measurement vector. The GPSR [50] targets the solution of the optimization problem

$$\min_{f'} \frac{1}{2}\|y - \Psi f'\|_2^2 + \lambda \|f'\|_1$$

for some $\lambda > 0$. The same problem appears e.g. in Basis Pursuit De-Noising [48]. The solution f' can be viewed as a function of the parameter λ and the optimization task as a decomposition of the measurement vector y into a signal part and error part:

$$y = \Psi f'(\lambda) + r(\lambda)$$

where $r(\lambda)$ is the residual error. When a bigger penalty is given to the l_1 norm of f' by using a bigger λ, a greater amount of the signal is moved to r and a more de-noised solution f' is obtained.

GPSR rewrites the aforementioned optimization problem as the quadratic program

$$\min_z c^T z + \frac{1}{2} z^T B z \quad \text{subject to} \quad z \geq 0$$

where $F(z) := c^T z + \frac{1}{2} z^T B z$ is the objective function (for B and c, see [50]). The solution is searched iteratively with refinements controlled by the gradient $\bigtriangledown F$ computed at each iteration. The computation of $\bigtriangledown F$ requires only matrix-vector products involving Ψ, Ψ^T. Therefore the GPSR is well suited for the class of applications where Ψ cannot be stored explicitly but there are fast transforms to compute Ψx and $\Psi^T x$ for a vector x.

Figueiredo et. al. [50] prove that the GPSR algorithm converges to the solution of the quadratic task at an R-linear rate [3] . Four different termination criterions are provided.

3.4.2 Greedy pursuits

The greedy strategy is to apply a sequence of locally optimal decisions to reach the globally optimal solution. In contrast, the optimization methods (Section 3.4.1) explicitly express the global optimum (i.e. the maximum reconstruction quality) and derive the steps to approach it. Greedy algorithms typically start with an empty solution and build the solution additively, whereas the optimization methods start with some initial solution which is iteratively refined [48]. Naturally, these conceptual differences lead to different practical properties; greedy algorithms are often empirically faster, but can easily loose their way by making a wrong decision. Optimization algorithms offer better theoretical guarantees but are computationally more complex.

In the description of greedy algorithms, we adhere to the terminology common in the field of signal approximation. In particular, the recovery task of finding f' such that $y = \Psi f'$ is referred to as the *decomposition* of y into the *atoms* of the *dictionary* Ψ. The setup is quite general, as it does not require the matrix Ψ to be a basis. Instead, Ψ is a (possibly redundant) collection of elements (*atoms*) which can be used to express the signal y. For example, Ψ may combine Fourier waveforms and spikes to provide an efficient sparse representation of y.

An important class of greedy algorithms used in compressive sampling is based on the ideas pioneered by Mallat and Zhang [51]. Their algorithm is called *matching pursuit* and since its birth in 1993, it has

[3] A sequence $\{x_k\}$ converges at an R-linear rate to L if there exists a sequence $\{\epsilon_k\}$ which converges linearly to zero and $|x_k - L| \leq \epsilon_k$ for all k.

given rise to many modifications such as *orthogonal matching pursuit*, *regularized orthogonal matching pursuit* and *compressive sampling matching pursuit*. Therefore, this class of algorithms is sometimes called *greedy pursuits* [52].

Matching pursuit

The matching pursuit (MP) algorithm was first introduced in [51] as a method of signal decomposition in terms of atoms of a given overcomplete dictionary. It guarantees recovery of the signal components which belong to the span of the dictionary, however, the guarantee is only asymptotical.

Algorithm 1 gives a pseudo-code of the matching pursuit. The algorithm works by additively refining the signal approximation and keeping track of the residual error. At the beginning of the algorithm, the residual error is set to the signal being decomposed - in the case of compressive sampling, to the measurement vector. At each step, a vector from the dictionary that has the highest correlation with the residual error is picked up (i.e. the vector that explains the residual error the best) and the signal approximation is updated by that correlation. In other words, the approximation is updated by repeatedly trying to explain the biggest portion of the residual error. In a practical implementation, the algorithm stops when the residual error is below a given threshold.

Data: $y \in R^k$, Ψ
Result: $f' \in R^n$
$x_i := 0$ for $i = 1, ...n$;
$r := y$;
while $\|r\| \leq \epsilon$ **do**
    /* Maximum correlation */
    $m := 0$;
    /* Index of the atom with the maximum correlation */
    $j := 0$;
    /* For all atoms... */
    for $i := 0; i < n; i++$ **do**
        /* Compute correlation of the atom and the residual error */
        $c_i := \langle r, \psi_i \rangle$;
        /* Look for the maximum correlation */
        if $|c_i| \geq m$ **then**
            $m := |c_i|$;
            $j := i$;
        end
    end
    /* Update the signal approximation */
    $f' := f' + c_j \psi_j$;
    /* Update the residual vector */
    $r := r - c_j \psi_j$;
end

Algorithm 1: Matching pursuit algorithm, [51]

Orthogonal matching pursuit

The orthogonal matching pursuit (OMP) was proposed by Pati et al. [53] as a refinement to the MP algorithm by Mallat and Zhang [51]. It aims at providing better guarantees for convergence; unlike MP, it guarantees convergence in at most n steps for an n-element dictionary. More precisely, the partial correctness statement is that at any given step, the algorithm has an optimal projection of the input signal onto the subset of the dictionary that was selected up to that time. At each step, one more atom of the dictionary is selected that has not been selected yet. The convergence statement is that the magnitude of the residual error goes to zero as the algorithm proceeds. Thus, at least after proceeding

through all dictionary entries, an optimal projection of the given signal on the subset of given dictionary is returned. Note that this notion of optimality does not say anything about the right choice of the dictionary subset. The algorithm minimizes the residual error, but can choose a wrong dictionary atom that prevents it from the right way of lowering the residual error.

Technically, the main difference with respect to MP is that OMP keeps track of the selected subset of the dictionary atoms and keeps the residual error orthogonal to it. This involves computing a least squares problem of size n in each step. Tropp and Gilbert [54] showed the following properties of OMP for the case of random measurement matrices.

Theorem 9 (Recovery by orthogonal matching pursuit, [54])**.** *Fix $\delta \in (0, 0.36)$ and S-sparse $f \in R^n$. Then construct a measurement matrix Ψ with +-1 entries drawn from symmetric Bernoulli distribution. Let $y = \Psi f$ be a measurement vector. If*

$$m \geq CS \ln(\frac{n}{\delta})$$

for some $C > 0$, orthogonal matching pursuit reconstructs the signal from measurement vector y exactly with probability at least $1 - \delta$.

Note that in comparison to similar theorems for basis pursuit (e.g. Theorem 5), this result is weaker in the sense that it guarantees a highly probable recovery for a fixed signal and a randomly constructed matrix. This means that one such random matrix is guaranteed to work with given probability only for the fixed signal. In contrast, Theorem 5 for BP first fixes the random matrix and then guarantees the exact recovery for a set of sufficiently sparse signals.

Regularized orthogonal matching pursuit

Needell and Vershynin [55] proposed a modification to the orthogonal matching pursuit which extends the procedure of selecting atoms from the overcomplete dictionary. OMP selects the atom that is most correlated to the residual error and has not been selected yet. Regularized OMP (ROMP) computes the correlation of the residual error to all remaining dictionary atoms and selects S correlations biggest in magnitude, where S is the sparsity level of the signal to approximate. Then the *regularization* step is performed: from the set of S atoms with the highest correlations, a subset with "comparable" magnitudes of correlations (see [55]) and maximum l_2 energy of the magnitudes is selected. The subdictionary used to express the signal is augmented by the whole selected set of atoms.

The goal of these modifications is to come with better guarantees of recovery than OMP:

Theorem 10 (Recovery by regularized orthogonal matching pursuit, [55])**.** *Let Ψ be a measurement matrix satisfying the restricted isometry conditions defined in [55]. Let $f \in R^n$ be an S-sparse vector and $y = \Psi f$ be the measurement vector. Then after at most n iterations, ROMP outputs a set of atoms I such that I is a superset of f's support and $|I| \leq 2S$. The signal f can be recovered exactly from measurements y by computing $(\Psi_I)^{-1}y$.*

The *restricted isometry conditions* defined in [55] are satisfied by random Gaussian, Bernoulli and partial Fourier matrices. Note that similar probability assumptions are required as in Theorem 5.

Compressive sampling matching pursuit

The compressive sampling matching pursuit (CoSaMP) was introduced very recently by Needell and Tropp [52]. As the name suggests, CoSaMP is derived from the matching pursuit. However, it combines the greedy approach with ideas from other types of recovery algorithms. First, it does a non-adaptive refinement of the signal estimate which is known from the iterative algorithms (Section 3.4.3). Second, it assumes the exact signal sparsity is known (compare to sublinear algorithms, Section 3.4.4) and uses it to parametrize the internals of the algorithm (see Algorithm 2).

Similarly to MP, CoSaMP keeps track of the residual error and tries to explain it in each iteration by applying Ψ^T. The obtained "signal proxy" is searched for large coefficients which are candidates for significant components of the estimated signal. A subset of the largest coefficients proportional to the

assumed sparsity level is added to the support of the signal estimate which is iteratively constructed. The signal estimate is then updated by computing the least squares estimate on the newly obtained support. Finally, the estimate is thresholded to contain only the number of coefficients corresponding to the signal sparsity.

Data: $y \in R^k$, Ψ, sparsity level m
Result: $x \in R^n$

`/* Intialize signal approximation */`
$x_i := 0$ for $i = 1, ...n$;

`/* Initialize residual error ax the measurement vector */`
$r := y$;

`/* Initialize support to an empty set */`
$S := \emptyset$;

while *stop criterion is not met* **do**

`/* Form signal proxy */`
$p := \Psi^T r$;

`/* Merge support with 2m largest components of p */`
$S := S \cup support(p, 2m)$;

`/* Update the approximation by computing least-squares on the support set */`
$x_S := LeastSquares(\Psi_S, y)$;
$x_{I-S} := 0$;;

`/* Take only m largest entries from x */`
$x := LargestEntries(x, m)$;

`/* Update the residual error */`
$r := y - \Psi x$;

end

Algorithm 2: CoSaMP algorithm, [52]

CoSaMP comes with uniform guarantees for all sensing matrices fullfiling restrictions on isometry constant. The guarantees bound l_2-norm of reconstruction error to be proportional to the amount of noise assumed in the signal model (measurements $y = \Psi x + \epsilon$) and to the amount of noise introduced by recovering a not exactly sparse signal.

3.4.3 Iterative thresholding

Iterative thresholding uses the Landweber iteration to converge to a signal approximation while repeatedly refining it by a thresholding function to ensure sparseness. The algorithm begins with an empty (zero) approximation. In each step, the residual error is computed and "inverted" by applying Ψ^T. This procedure yields an estimate of the residual error expressed in Ψ. To explain this error, the sparse approximation is updated by this estimate and consecutively, the thresholding is applied to filter significant coefficients. This approach was analyzed by Daubechies et. al. [56].

Data: $y \in R^k$, Ψ
Result: $x \in R^n$
$x_i := 0$ for $i = 1, ...n$;
while *stop criterion is not met* **do**
    `/* Update the coefficient vector */`
    $x := x + \Psi^T(y - \Psi x)$;
    `/* Perform thresholding */`
    $x := T(x)$;
end

Algorithm 3: Iterative thresholding, [56]

3.4.4 Sublinear algorithms

Gilbert et al. [57] proposed the *chaining pursuit* algorithm with recovery time sublinear in the length of the reconstructed signal. Their approach is based on the theory of dimensionality reduction (see e.g. [58]). In short, it is known that it is possible to reduce a set of points in $\mathbb{R}^n$ which have non-zero coordinates in at most S dimensions to a subspace of $O(S \log^2(n))$ dimensions, provided that some distortion can be accepted. Practically, this reduction is achieved by applying a specially constructed, structured sampling matrix. The recovery algorithm works by extracting only the significant coefficients of the signal, i.e. works in time proportional to the signal sparsity.

Chaining pursuit provides uniform guarantees of recovery for approximately sparse signals:

Theorem 11 (Recovery with chaining pursuit, [57])**.** *With probability at least* $1 - O(d^{-3})$*, the chaining pursuit measurement operator* Ψ *has the following property. Suppose that* f *is a* d*-dimensional signal whose best* m*-term approximation with respect to* l_1 *norm is* f_m*. Given the measurements* $y = \Psi f$ *of size* $O(m \log^2 d)$ *and the measurement matrix* Ψ*, the chaining pursuit algorithm produces a signal estimate* f' *with at most* m *nonzero entries. The output* f' *satisfies*

$$\|f - f'\|_{l_1} \leq C(1 + \log m)\|f - f_m\|_{l_1}$$

In particular, if $f_m = f$*, then also* $f' = f$*. The time cost of the algorithm is* $O(m \log^2(m) \log^2(d))$*.*

In comparison to the basis pursuit approach, chaining pursuit requires a special design of the sampling matrix. At the time of sampling, BP does not require to know the sparsity basis of the signal; chaining pursuit must perform the encoding by applying the sparse transformation at the sampling time. Chaining pursuit requires slightly more measurements, in concrete, $O(n^2 \log n)$ compared to $O(n \frac{n}{S})$ of BP with random Gaussian matrix.

Recently, a similar algorithmic approach was proposed by the same authors as *HHS Pursuit* [59]. HHS Pursuit uses chaining pursuit as an optional preprocessing step.

3.4.5 Time complexity of recovery algorithms

Consider a recovery of n length, S-sparse signals from m measurements. Table 3.1 summarizes the time complexity of the recovery if no special properties of the operators involved are exploited. By employing fast transforms which are well known for many operators used in the recovery (such as the fast Fourier transform, fast wavelet transform), the time complexity can be further lowered. Other important criterions for quantitative comparison of recovery algorithms are spatial complexity and accuracy. We refer the reader to [57] for a similar comparison involving the reconstruction accuracy.

3.5 Bases with low coherency

The theorems introduced in Section 3.3.4 can be practically used only if concrete pairs of incoherent bases are known. We briefly visit some of the frequently used ones [1]. The representatives encompass two perfectly incoherent pairs, the canonical (time) basis vs. Fourier basis and noiselet basis vs. wavelet

Recovery task	Technique	Time complexity
Basis pursuit with equality constraint	Linear programming	$O(m^2 n^{\frac{3}{2}})$
Basis pursuit with quadratic constraint	Second order cone programming	$O(n^3)$
Basis pursuit with Dantzig selector	Linear programming	$O(m^2 n^{\frac{3}{2}})$
GPSR	Quadratic programming	$O(n^2)$ per iteration
Matching pursuit	Greedy algorithm	$O(mn)$ per iteration
Orthogonal matching pursuit	Greedy algorithm	$O(Smn)$
Regularized OMP	Greedy algorithm	$O(Smn)$
CoSaMP	Greedy algorithm	$O(mn)$
Iterative thresholding	Iterative algorithm	$O(mn)$ per iteration
Chaining pursuit	Combinatorial algorithm	$O(m \log^2(m) \log^2(n))$

Figure 3.1: Time complexity of selected recovery algorithms. Signal length is n, signal sparsity S and measurement count m.

bases. Both of them are interesting because they are perfectly or nearly perfectly incoherent to well established domains of time-frequency analysis. We also mention the interesting case of random matrices which are highly (but not perfectly) incoherent to any representation basis of choice. This has both implementational and algorithmic consequences: only one sampling algorithm has to be implemented while the search for the ideal decorrelation basis can evolve or the basis can be adaptively changed during the recovery.

3.5.1 Canonical spike basis and Fourier basis

Canonical spike basis and Fourier basis are known to be maximally incoherent.

Canonical spike basis

The canonical spike basis consists of vectors having all coefficients set to zero except one (the spike). Such a basis can be conveniently used to represent a signal using samples in the time or space domain. Formally, a spike basis can be defined using the Dirac delta functions:

Definition 9 (Dirac delta function, [60])**.** The Dirac delta function on $\mathbb{R}$ is

$$\delta(x) = \begin{cases} 1, & x = 0 \\ 0, & x \neq 0 \end{cases}$$

Note that we are using simplified definition of the Dirac function which is originally defined as a limit of rectangular functions (see e.g. [61]).

Definition 10 (Canonical (spike) basis, [1])**.** We say that $\Psi = \begin{bmatrix} \psi_1 & \psi_2 & ... & \psi_n \end{bmatrix}$ is a spike basis if

$$(\psi_k)_i = \delta(i - k)$$

Fourier basis

We are often interested in converting a signal represented in the time domain to a frequency domain, or vice versa. This task is referred to as the time-frequency analysis. Fourier analysis is one of the well known methods of the time-frequency analysis. It represents the analyzed signal in terms of harmonic sinusoidal functions. The operation of transforming a signal f to the frequency domain using the Fourier analysis is called *Fourier transform*. In its most general form, the Fourier transform concerns functions of real arguments (i.e. $f(t)$) and is defined as [61]:

$$S(F) = \int_{-\infty}^{\infty} f(t) \cdot e^{-i2\pi Ft} dt$$

The expression above defines a complex number describing the amplitude and phase of the frequency F in the frequency representation of $f(t)$. In the digital world, we are concerned with the *discrete Fourier transform* (DFT) of a discrete signal $f[t]$. The resulting frequency representation $S[F]$ is discrete as well and has the following form:

$$S[F] = \sum_{t=0}^{N-1} f[t] \cdot e^{-i2\pi \frac{F}{N} t}$$

Definition 11 (Fourier basis, [61])**.** We say that a basis $\Psi = \begin{bmatrix}\psi_1 & \psi_2 & ... & \psi_n\end{bmatrix}$ consisting of basis vectors

$$(\psi_j)_t = n^{-\frac{1}{2}} e^{i2\pi jt/n}$$

is called a Fourier basis.

3.5.2 Wavelets and noiselets

Wavelets

Wavelets [61] provide a way to perform time-frequency analysis using families of wavelet functions parametrized by shift (translation) and scale. Compared to Fourier analysis, this approach yields better results in analysis of non-periodic signals and signals with local discontinuities and changes. Given a mother wavelet $\psi(t)$, a family of wavelet functions $\psi_{a,b}(t)$ is derived by shifting and scaling of ψ [61]:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}} \psi\left(\frac{t-b}{a}\right)$$

where a denotes scale and b denotes shift.

The operation of decomposing a real signal $f(t)$ [4] to the wavelet domain is called a *wavelet transform*. In its most general form (*continuous wavelet transform*), it is defined as [61]:

$$CWT_{a,b}(f) = \frac{1}{\sqrt{a}} \int_R \psi\left(\frac{t-b}{a}\right) f(t)\, dt$$

where a, b are scale and shift, respectively. To be able to compute the wavelet transform practically for sampled signals, the wavelet family is discretized. For $m, n \in \mathbb{Z}, a_0 > 1, b_0 > 0$, the discretized child wavelets are obtained as [61]

$$\psi_{m,n}(t) = a_0^{-m/2} \psi(a_0^{-m} t - n b_0)$$

where m denotes scale and n denotes shift. The *discrete wavelet transform* takes the form [61]

$$DWT(f) = \sum_m \sum_n \langle \psi_{m,n} f \rangle$$

The discrete wavelet transform can be effectively computed by exploiting the multiresolution structure of wavelets and designing multiscale filter banks [62]. For example, the discrete Haar wavelet transform can be computed in $O(n)$ time, where n is the signal length.

There is a multitude of wavelet families with varying complexity of design and implementation. The first wavelet was created by a Hungarian mathematician Alfred Haar in 1909 and is therefore called the *Haar wavelet*. It is the simplest possible wavelet and is also considered as a special case of the Daubechies wavelet [63]. The JPEG 2000 standard [42] employs the Cohen-Daubechies-Feauveau wavelet [63]. As the detailed survey of wavelet families is beyond the scope of this work, we refer the reader e.g. to [63] for a more elaborate discussion on the properties of different wavelet families.

[4] More precisely, we require $f(t) \in L_2(\mathbb{R})$

Noiselets

Noiselets [64] are functions designed to be totally uncompressible using the Haar wavelet analysis. They can be shown to generate orthonormal bases for the spaces of Haar multiresolution analysis and feature fast transform algorithms. The family of noiselets is constructed on the interval $[0, 1)$as follows:

$$\begin{aligned} f_1(x) &= \chi_{[0,1)}(x) \\ f_{2n}(x) &= (1-i)f_n(2x) + (1+i)f_n(2x-1) \\ f_{2n+1}(x) &= (1+i)f_n(2x) + (1-i)f_n(2x-1) \end{aligned}$$

Here, $\chi_{[0,1)}(x) = 1$ on the whole definition interval $[0, 1)$. The construction of noiselets can be extended to $\mathbb{R}$. It can be shown that

Theorem 12 (Orthogonality of noiselets, [64])**.** *The set $\{f_j | j = 2^N, ..., 2^{N+1} - 1\}$ is an orthogonal basis of the vector space V_{2^N}, which is a space of all possible approximations at the resolution 2^N of functions in $L^2(\mathbb{R})$.*

The theorem says that a given set of noiselets forms an orthogonal basis of the space of all approximation functions we get by computing the Haar wavelet decomposition with a given resolution. This means that the same class of functions can be expressed by noiselets and Haar wavelets. Interestingly, Haar wavelet coefficients of noiselets are flat up to the finest scales (see [64]), meaning that noiselets themselves cannot be compressed in the Haar basis. In terms of compressive sampling, this is described by the perfect incoherence property.

Noiselets can be also expressed using the means of linear algebra. Starting with a 1×1 matrix N_1, a sequence of noiselet matrices $N_1, N_2, N_4, ..., N_{2^m}$ of sizes 1×1, 2×2, 4×4, ..., $2^m \times 2^m$, respectively, is generated. The rows of the N_n matrix are the noiselets which form an orthonormal basis for the space $\mathbb{R}^n$.

Definition 12 (Noiselet matrix)**.** The $n \times n$ noiselet matrix N_n is defined recursively as

$$\begin{aligned} N_n(k, *) &= \frac{1}{2}\left[(1-i, 1+i) \otimes N_{n/2}(\lfloor \tfrac{k}{2} \rfloor, *)\right] \quad \text{for k=0,2,4,...,n-2} \\ N_n(k, *) &= \frac{1}{2}\left[(1+i, 1-i) \otimes N_{n/2}(\lfloor \tfrac{k}{2} \rfloor, *)\right] \quad \text{for k=1,3,...,n-1} \end{aligned}$$

starting with $N_1 = \left[1\right]$.

See Chapter 4 for a matrix-based proof of the perfect incoherence between the noiselets and Haar wavelets. Concerning other types of wavelets, [1] gives estimates of the incoherence for the Daubechies-4 and Daubechies-8 wavelets, which are also constant and low: 2.2 and 2.9, respectively.

One possible way of implementing a fast noiselet transform is to take the Cooley - Tukey approach known from the Fast Fourier Transform [60].

3.5.3 Random basis and any fixed basis

So far, we have given examples of incoherence between particular types of sampling and representation bases. However, it can be shown that a random sampling basis is highly incoherent with *any* fixed representation basis:

Theorem 13 (Incoherence of a random and any fixed basis, [1])**.** *Let Ψ be a random basis of the space $\mathbb{R}^n$ (created by sampling randomly and uniformly points on the unit sphere in $\mathbb{R}^n$) and Φ be any n-dimensional basis. Then the incoherence $C(\Psi, \Phi)$ is approximately $\sqrt{2 \log n}$.*

The theorem says that if we take samples by observing the first K coefficients of the signal representation in a randomly chosen n-dimensional basis, we get incoherence guarantees independent of

the representational basis we use in the recovery. In practical implementations, the concept of the downsampling in a random basis can be further simplified. Instead of creating the basis and ensuring its orthogonality, a sampling matrix with coefficients sampled directly from a Gaussian or Bernoulli distribution can be used directly. See Theorem 5 in Section 3.3.3.

Chapter 4

On the incoherence of noiselet and Haar bases

In Chapter 3, we have seen that pairs of bases which exhibit low coherency are important in compressive sampling. The more incoherent the bases are, the better compression ratios can be achieved. We have also pointed out that noiselets [64], introduced by Coifman, Geshwind and Meyer in 2001, are perfectly incoherent to the Haar wavelet basis. Because noiselets come with a fast transform algorithm, they have attracted interest as a sampling basis in compressive sampling.

In this chapter, a simple proof of the incoherence between the noiselet and Haar bases is given. By introducing the proof we extend the theoretical background presented in Chapter 3 and allow the reader to understand one of the important properties used in the remainder of the text. Unlike the proof currently available in the literature [64], the incoherence is shown in the language of simple linear algebra. A new, elegant recursive equation for noiselets is derived which allows for a Kronecker-product based proof.

4.1 Preliminaries

4.1.1 General definitions

Definition 13 (Kronecker product, [65])**.** Let A be an $m \times n$ matrix, let B be a matrix of an arbitrary size. The Kronecker product of A and B is

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

The Kronecker product is a bilinear and associative operator. It is not generally commutative. It can be mixed with a standard maxtrix multiplication in the following way

$$(A \otimes B)(C \otimes D) = AC \otimes BD$$

whenever the products AC, BD exist. This property is sometimes called the *mixed product property*. For further information on the Kronecker product, we refer the reader to e.g. [65].

Definition 14 (Matrix row/column notation)**.** Let A be a $m \times n$ matrix. By $A(k, *)$ we denote the (row) vector

$$[A(k,1), A(k,2), ..., A(k,n)]$$

and similarly, $A(*, l)$ denotes the (column) vector

$$[A(1,l), A(2,l), ..., A(m,l)]^T$$

4.1.2 Noiselets

Noiselets [64] are functions designed to be totally uncompressible using the wavelet analysis. The family of noiselets is constructed on the interval $[0, 1)$ as follows:

$$\begin{aligned} f_1(x) &= \chi_{[0,1)}(x) \\ f_{2n}(x) &= (1-i)f_n(2x) + (1+i)f_n(2x-1) \\ f_{2n+1}(x) &= (1+i)f_n(2x) + (1-i)f_n(2x-1) \end{aligned}$$

Here, $\chi_{[0,1)}(x) = 1$ on the definition interval $[0, 1)$ and 0 otherwise.. It can be shown that

Theorem 14 (Orthogonality of noiselets, [64]). *The set $\{f_j | j = 2^N, ..., 2^{N+1} - 1\}$ is an orthogonal basis of the vector space V_{2^N}, which is a space of all possible approximations at the resolution 2^N of functions in $L^2([0, 1))$.*

4.1.3 Haar wavelets

The Haar wavelet basis can be described by a real square matrix. For our purposes, it is advantageous to recursively build the $n \times n$ Haar wavelet matrix H_n using the Kroneckner product [66]:

$$H_n = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{n/2} \otimes (1, 1) \\ I_{n/2} \otimes (1, -1) \end{bmatrix}$$

The iteration starts with $H_1 = \begin{bmatrix} 1 \end{bmatrix}$. The normalization constant $\frac{1}{\sqrt{2}}$ ensures that $H_n^T H_n = I$. Haar wavelets are the rows of H_n.

4.2 Matrix construction of noiselets

First we extend and discretize the noiselet functions.

Definition 15 (Extended noiselets). The extensions of noiselets to the interval $[0, 2^m - 1]$ sampled at points $0, 1, ..., 2^m - 1$ is the series of functions $f_m(k, l)$

$$\begin{aligned} f_m(1, l) &= 1 \quad \text{for } l = 0, ..., 2^m - 1 \\ &= 0 \quad \text{otherwise} \\ f_m(2k, l) &= (1-i)f_m(k, 2l) + (1+i)f_m(k, 2l - 2^m) \\ f_m(2k+1, l) &= (1+i)f_m(k, 2l) + (1-i)f_m(k, 2l - 2^m) \end{aligned}$$

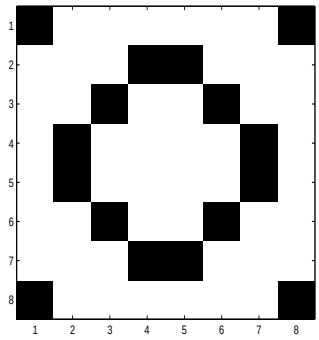
where m denotes the range of extension, $k = 1, ..., 2^{m+1}$ is the function index and $l = 0, ..., 2^m - 1$ is the sample index.

Starting with a 1×1 matrix N_1, a sequence of noiselet matrices $N_1, N_2, N_4, ..., N_{2^m}$ of sizes 1×1, 2×2, 4×4, ..., $2^m \times 2^m$, respectively, is generated. The rows of the N_n matrix are the noiselets which form an orthonormal basis for the space $\mathbb{R}^n$.

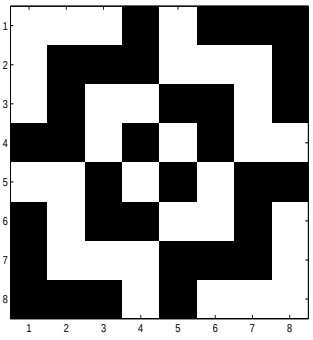
Definition 16 (Matrix recursion for noiselets). The $n \times n$ noiselet matrix N_n is defined recursively as

$$\begin{aligned} N_n(k, *) &= \frac{1}{2} \left[(1-i, 1+i) \otimes N_{n/2}(\lfloor \tfrac{k}{2} \rfloor, *) \right] \quad \text{for k=0,2,4,...,n-2} \\ N_n(k, *) &= \frac{1}{2} \left[(1+i, 1-i) \otimes N_{n/2}(\lfloor \tfrac{k}{2} \rfloor, *) \right] \quad \text{for k=1,3,...,n-1} \end{aligned}$$

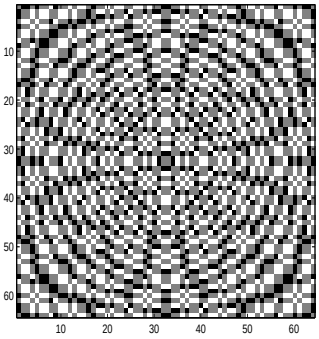
starting with $N_1 = \begin{bmatrix} 1 \end{bmatrix}$.



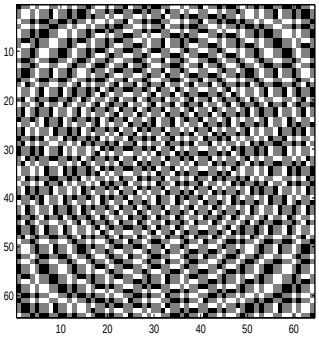
(a) Real part of 8x8 noiselet matrix



(b) Imaginary part of 8x8 noiselet matrix



(c) Real part of 64x64 noiselet matrix



(d) Imaginary part of 64x64 noiselet matrix

Figure 4.1: Noiselet matrix: graphical view. In figures (a) and (b), the black and white colors denote values of -0.25 and 0.25 respectively. In figures (c) and (d), the black, gray and white colors denote values of -0.125, 0 and 0.125 respectively.

Theorem 15 (Relation of the noiselet matrix to the extended noiselets)**.** *Let $m > 0$. The noiselet matrices $N_1, N_2, N_4, ..., N_{2^m}$ relate to the series of f_m functions as*

$$N_n(k,l) = f_m(n+k, \frac{2^m}{n}l) \quad \textit{for } k,l = 0,...,n-1$$

Proof. Let $m > 0$ be fixed. For $n = 1$

$$N_1(0,0) = f_m(1,0) = 1$$

By induction, for a matrix of size $n = 2^p, p = 1, ..., m$, its basis vector $k = 0, 2, 4, ..., n-2$ and vector indices $l = 0, ..., \frac{n}{2} - 1$

$$N_n(k,l) = (1-i)N_{n/2}(\lfloor\frac{k}{2}\rfloor, l) = (1-i)f_m(\frac{n}{2} + \frac{k}{2}, \frac{2^m}{\frac{n}{2}}l) = f_m(n+k, \frac{2^m}{n}l)$$

For the same n, k and $l = \frac{n}{2}, ..., n-1$

$$N_n(k,l) = (1+i)N_{n/2}(\lfloor\frac{k}{2}\rfloor, l - \frac{n}{2}) = (1+i)f_m(\frac{n}{2} + \frac{k}{2}, 2\frac{2^m l}{n} - 2^m) = f_m(n+k, \frac{2^m}{n}l)$$

To see this, observe that f_m is zero outside the interval $[0, 2^m - 1]$ and therefore, the first half of samples of $f_m(k,l)$ are defined exclusively by the expression $(1 \pm i)f_m(k, 2l)$ whereas the second half of the samples are defined exclusively by $(1 \pm i)f_m(k, 2l - 2^m)$.

The situation is analogical for $k = 1, 2, ..., n-1$. □

Specially, the noiselet matrix N_n for $n = 2^m$ can be found as the "tail" of the function series f_m. Indeed, the expression in Theorem 15 becomes $N(k,l) = f_m(n+k, l)$ for $n = 2^m$.

4.3 Incoherence of noiselets and Haar

Definition 17 (Perfect incoherency)**.** Two bases A, B of $\mathbb{C}^n$ are perfectly incoherent if the matrix $C = AB^T$ is "flat", i.e.

$$\exists c \in \mathbb{R} : |C(k,l)| = c \quad \text{for all } 0 \leq k, l \leq n-1$$

We show the perfect incoherence of the noiselet and Haar basis. It will save us some technical work to define a "twisted" noiselet basis as

Definition 18 (Twisted noiselets)**.** The $n \times n$ noiselet matrix $\hat{N}_n$ is defined recursively as

$$\hat{N}_n(k,*) = \frac{1}{2}\left[\hat{N}_{n/2}(\lfloor \tfrac{k}{2} \rfloor, *) \otimes (1-i, 1+i)\right] \quad \text{for k=0,2,4,...,n-2}$$
$$\hat{N}_n(k,*) = \frac{1}{2}\left[\hat{N}_{n/2}(\lfloor \tfrac{k}{2} \rfloor, *) \otimes (1+i, 1-i)\right] \quad \text{for k=1,3,...,n-1}$$

starting with $\hat{N}_1 = \left[1\right]$.

Compared to the definition of N, we only changed the order of operands in the Kronecker product. We can convince ourselves that

Theorem 16 (Equivalency of twisted noiselets and noiselets)**.** *For $n = 2^m$, the bases N_n, $\hat{N}_n$ consist of the same set of basis vectors.*

Proof. Indeed, we can write $\hat{N}_n = P_n N_n$ where P is a permutation matrix defined as

$$P(k,*) = P_{n/2}(\lfloor \frac{k}{2} \rfloor, *) \otimes (1,0) \quad \text{for } k = 0, 2, 4, ..., n-2$$
$$P(k,*) = P_{n/2}(\lfloor \frac{k}{2} \rfloor, *) \otimes (0,1) \quad \text{for } k = 1, 3, ..., n-1$$

starting with $P = [1]$.

The claim holds for $n = 1$. For $n = 2, 4, 8, ..., 2^m$,

$$P_n N_n(k,l) = P_n(k,*) N_n^T(*,l)$$

as it can easily be shown that N_n is symmetric. Using the recurrent equations for P_n and N_n and applying the mixed product rule, we get for $l = 0, 2, 4, ..., n-2$

$$P_n N_n(k,l) = (1-i) P_{n/2}(\lfloor \frac{k}{2} \rfloor, *) N_{n/2}(*, \frac{l}{2})$$

$$P_n N_n(k,l) = (1+i) P_{n/2}(\lfloor \frac{k}{2} \rfloor, *) N_{n/2}(*, \frac{l}{2})$$

where $k = 0, 2, 4, ..., n-2$ and $k = 1, 3, ..., n-1$, respectively. By applying the induction we get

$$P_n N_n(k,*) = (1-i, 1+i) \otimes \hat{N}_{n/2}(\lfloor \frac{k}{2} \rfloor, *)$$

for even l indices. This situation for odd l indices is similar. □

Now the main result can be shown.

Theorem 17 (Perfect incoherency of Haar and noiselets)**.** *Let $n = 2^m, m > 0$. Let N_n be the noiselet matrix of size $n \times n$ and let H_n be the Haar matrix of size $n \times n$. Assuming the bases are normalized such that $H_n^T H_n = I$ and $N_n^T N_n = nI$, the coherence matrix $C = H_n N_n^T$ has all entries of magnitude 1, i.e. $|C(k,l)| = 1$ for all $0 \leq k, l \leq n-1$.*

Proof. It is trivial to show the claim for the case of $n = 1$, as

$$H_1 N_1 = \begin{bmatrix}1\end{bmatrix} \cdot \begin{bmatrix}1\end{bmatrix} = \begin{bmatrix}1\end{bmatrix}$$

For $n = 2^m, m > 1$, the incoherence is shown by induction. Suppose we know the perfect incoherence holds for $\frac{n}{2}$ and we want to show it for n. In the induction step, we use the iterative construction of the Haar matrix by means of Kronecker product. By computing the product

$$H_n \hat{N}_n^T = H(N_n^T P_n^T) = (H_n N_n^T) P_n^T$$

we will still be able to conclude on magnitude of the elements of $(H_n N_n^T)$, since the permutation matrix does not change the magnitudes.

The product $H_n \hat{N}_n^T$ can be computed per-column; we take the j-th column of $\hat{N}_n^T$, $j = 0, 2, 4, ..., n-2$ and transform it by H_n, getting

$$H_n \hat{N}_n^T(*, j) = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{n/2} \otimes (1,1) \\ I_{n/2} \otimes (1,-1) \end{bmatrix} \cdot \frac{1}{\sqrt{2}} \left[\hat{N}_{n/2}^T(*, \tfrac{j}{2}) \otimes (1-i, 1+i)^T \right]$$

Note the altered normalization factor of noiselets. Now the mixed product property can be applied to get

$$\frac{1}{2} \begin{bmatrix} H_{n/2} \hat{N}_{n/2}^T(*, \frac{j}{2}) \otimes (1,1) \begin{bmatrix} 1-i \\ 1+i \end{bmatrix} \\ I_{n/2} \hat{N}_{n/2}^T(*, \frac{j}{2}) \otimes (1,-1) \begin{bmatrix} 1-i \\ 1+i \end{bmatrix} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} H_{n/2} \hat{N}_{n/2}^T(*, \frac{j}{2}) \otimes 2 \\ I_{n/2} \hat{N}_{n/2}^T(*, \frac{j}{2}) \otimes -2i \end{bmatrix}$$

By induction, it follows that

$$|H_{n/2} \hat{N}_{n/2}^T(i, \frac{j}{2})| = 1$$

$$|I_{n/2} \hat{N}_{n/2}^T(i, \frac{j}{2})| = 1$$

for $i = 1, ..., \frac{n}{2}$. The Kronecker multiplication is only by entries with magnitude 2, thus the resulting magnitudes are $\frac{1}{2} * 2 = 1$. The proof is parallel for $j = 1, 3, ..., n-1$. □

Chapter 5

Compressibility of performance signals

Compressive sampling is designed to work with signals that are sparse in a given representation basis. When an exact sparsity level cannot be determined due to the nature of the signal, many flavors of compressive sampling can be shown to behave in a stable way provided the signal is approximately sparse. Thus, a characterization of the signal compressibility has to be known in order to predict behaviour of the compression algorithms and to tune compressive sampling for a particular application.

In this chapter, we measure a set of real performance signals on a wide spread processor platform and derive estimates of their sparsity in selected bases. We do this empirically by analyzing the representation of the acquired signals in the sparsity domain.

5.1 Data collection

The environment for collection of performance signals can be divided into the workload layer (a set of programs that are measured), the hardware layer (the execution platform which is measured) and the software layer (a set of programs that enable to acquire and store the measurement data). After choosing a concrete instance of this measurement stack, we select a range of performance signals to be measured and perform the actual measurements.

5.1.1 Software environment

The interface for processor performance monitoring can be different not only across the multitude of processor models, but also across the implementations of one processor model. Therefore it is advantageous to use a software interface which provides a reasonable abstraction over the hardware-dependent layers.

In the following experiments, we use the Linux® operating system and the performance monitoring project *perfmon* [15]. Perfmon defines a logical view of the performance monitoring interface (also called PMU, performance monitoring unit) which consists of a set of logical register pairs. Each pair consists of:

- PMC - Performance monitoring control register
- PMD - Performance monitoring data register

As the name suggests, a PMC is used to control the content of the corresponding PMD. Depending on the underlying hardware, the registers can also be decoupled, so that one PMC controls a vectors of PMDs or vice versa.

All PMCs and PMDs are equally wide (64 bits). The mapping from the logical PMCs and PMDs to the concrete hardware interface is done internally, effectively hiding the differences in PMU implementations from the user. There are two modes of performance monitoring available, *per-thread monitoring*

and *system-wide monitoring*. Per-thread monitoring preserves the exclusive state of the PMU for each thread across task switches, whereas a system-wide session measures the performance events for a given set of logical processors, such that the events generated by all tasks are aggregated. In both modes, it is possible to filter the events generated at the kernel or user level.

The functionality provided by *perfmon* kernel libraries is encapsulated in the user-space command line utility *pfmon*. In order to obtain the processor signal, pfmon can be set up to sample a given set of counters with a given frequency. There are two ways how the sampling can be done: the *time interval printing* or *event sampling* functionality.

Using the time interval printing, the values of selected PMDs are printed or stored periodically in a given interval. This is the most straightforward approach which is also used in the experiments in this chapter. However, the event sampling could also be used for this purpose. It samples down a set of event counters each time a given counter exceeds a specific value. Provided there is a performance event that occurs regularly, such an event can be used to trigger the sampling of the counters.

5.1.2 Workload

We used a subset of SPEC CPU2006 benchmark suite [67] as the workload for our measurements. The benchmark subset was selected to include different types of workload in terms of application domain and character of CPU usage. In particular, we chose compilation, video encoding, ray tracing and fluid dynamics as workload types. These are summarized in Table 5.1.

Integer benchmarks	
403.gcc	C Language optimizing compiler
464.h264ref	Video compression
Floating point benchmarks	
437.leslie3d	Computational Fluid Dynamics
453.povray	Computer visualization

Table 5.1: SPEC CPU2006 benchmarks selected as the workload.

5.1.3 Hardware platform

We gathered performance information on the Intel® Core 2 Duo 2.33 GHz processor [10]. The processor consists of two execution cores, each core has its own execution resources and execution state. Both cores share the second level cache and bus. Figure 5.1 gives an overall scheme of the processor.

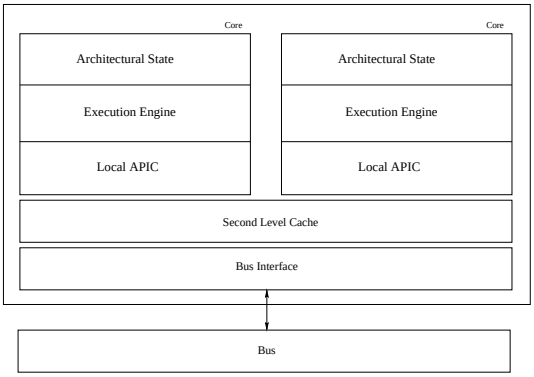


Figure 5.1: Overall architecture of the Intel® Core 2 Duo chip used in the experiments. The scheme is based on [10]. In a dual-core system, each of the cores has its exclusive execution state and execution resources. The bus interface and second level cache are shared among the cores.

The processor uses out-of-order instruction execution. This means that performance of the pipeline is strongly influenced by the workload. Consequently, monitoring of the events related to the pipeline performance is of natural interest. The processor features a two-way cache architecture (compare to Section 2.1.2). Each core has its own L1 cache (separate for instructions and data) and TLB. The L2 cache is shared between the cores.

Performance monitoring

For each logical processor, the Intel® Core 2 architecture defines a set of register pairs which provide an interface to the performance monitoring capabilities (compare to Section 5.1.1):

- IA32_PERFEVTSELx (control register)
- IA32_PMCx (data register)

The CPUID mechanism [10] can be used to determine the size and count of available performance monitoring registers which can differ across processor implementations. The control register specifies the content and behaviour of the respective counter register. It allows the programmer to set the event and its particular alternate to be counted, filter the events according to the privilege level of the instruction and generate an interrupt in case of counter overflow.

5.1.4 Performance signals

A complete list of performance events available on the Intel® Core 2 platform can be found in [10]. We selected a subset of the available performance events as the data source for the experiments, see Table 5.2. The events were selected such that they reflect performance of important infrastructure components of the processor.

Event group	Event code and mask	Event description
		Architectural events
General events	INST_RETIRED:ANY_P	Retired instructions
	CPU_CLK_UNHALTED.REF	Reference cycles when the core is not halted
		Caches and TLBs
L1 cache	L1D_REPL	Cache lines allocated in the L1 data cache (L1 cache misses)
	L1I_MISSES	Instruction Fetch Unit misses
L2 cache	L2_LINES_IN.CORE	L2 cache misses
	L2_M_LINES_IN	L2 cache line modifications
	L2_M_LINES_OUT	Modified lines evicted from the L2 cache
Instruction TLB	ITLB_MISSES	ITLB misses
Data TLB	DTLB_MISSES:ANY	Memory accesses that missed the DTLB
		Floating point unit
Floating point operations	MUL	Multiply operations executed
	DIV	Divide operations executed
		Pipeline and internal execution processes
Pipeline stalls	RESOURCE_STALLS.ANY	Resource related stalls
Branches	BR_MISP_EXECUTED	Mispredicted branch instructions executed
Load/store mutual blocking	SB_DRAIN_CYCLES	Cycles while stores are blocked due to store buffer drain
Memory disambiguation	MEMORY_DISAMBIGUATION.RESET	Memory disambiguation reset cycles
		Bus
Bus transactions	BUS_TRANS_MEM.CORE	Memory bus transactions
		Other
Interrupts	HW_INT_RCV	Hardware interrupts received

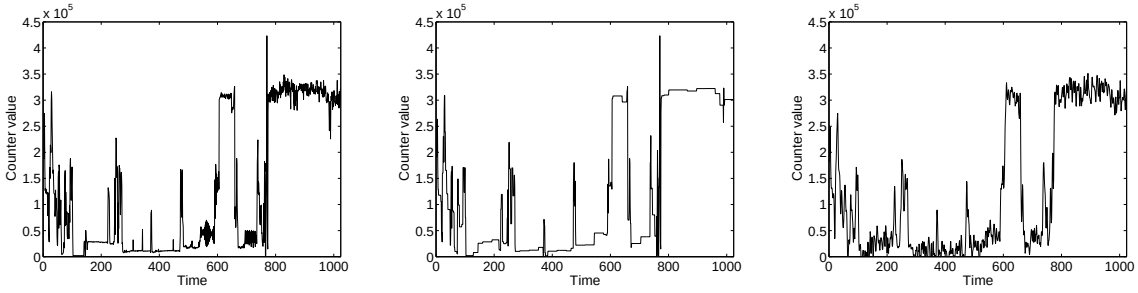
Table 5.2: List of evaluated performance events on the Intel® Core 2 architecture.

Measurement frequency

The performance counters were sampled periodically every 10ms.

5.2 Experimental analysis

Sparsity of the input signal in a chosen representation basis is the key assumption of compressive sampling (see Section 3.3). In particular, the degree of sparsity influences the number of samples that have to be taken in the sampling phase, respectively, it determines the probability that an exact reconstruction from a given number of samples occurs. In the following analysis, we perform very similar steps to the initial phases of classic compression schemes, i.e. we compute the signal transformation and examine coefficients in the basis that is expected to provide a concise representation. We shall be interested in determining to what extent non-significant coefficients can be discarded while the important information carried by the signal is preserved. However, keep in mind that the compressive sampling algorithm itself does not work by examining sparse representation coefficients in this way and does not perform any operations that would need to know what coefficients to discard.



(a) Original signal

(b) Recovery from 1/8 of Haar wavelet coefficients

(c) Recovery from 1/8 of DCT coefficients

Figure 5.2: Intel® Core 2: number of retired mispredicted branch instructions, sampled while running SPEC CPU2006 403.gcc workload. The original signal consists of 1024 samples. The signals in figures (b) and (c) were recovered from 128 most significant coefficients in given representation basis.

5.2.1 General setup

As we do not assume the signals to be exactly sparse, it is not possible to directly estimate the number of non-zero coefficients for a given signal in a given representational basis. We expect the energy of the signals to be partially spread out over many coefficients which are, under the assumption of compressibility, not significant, i.e. such coefficients do not carry substantial information about the signal. Therefore instead of an exact sparsity level, we estimate the number of *significant* coefficients in the representational domain. Figure 5.3(a) illustrates the coefficient magnitudes used in a representation of a real signal. It shows that the signal is approximately sparse when represented in both the DCT and Haar domain, for example in both representations the top ten coefficient have much higher values than all subsequent coefficients.

A good basis for a given signal is one that concentrates most of the signal's energy into as few coefficients as possible. Therefore the task of sparsity detection can be reformulated as finding a threshold for classifying high-valued coefficients. We assume that such a threshold is signal and representation dependent. It also depends on the desired level of accuracy.

In order to determine the threshold and its impact on the reconstruction quality, we introduce the following procedure. First the signal is transformed to the representational basis. The representational coefficients are examined to detect the threshold. Next, a thresholding operation on the representation coefficients is applied and the signal is recovered from the thresholded representation coefficients. After the recovery, similarity between the original and the recovered signal is measured. More precisely, we use the following entities:

- Input signal $f \in \mathbb{R}^n$
- Representation basis Ψ

- Transformation function $T : \mathbb{R}^n \to \mathbb{R}^n$
- Inverse transformation function $T^{-1} : \mathbb{R}^n \to \mathbb{R}^n$
- Thresholding function $C : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$
- Similarity metric $S : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$

A more detailed description of the entities follows:

Transformation functions. The function T corresponding to the representation basis Ψ converts a given input signal f into its representation in Ψ. Note that the representation vector $T(f)$ contains coefficients whose meaning and layout is given by the particular transformation computed and is likely to differ for different transformations. For instance, a wavelet transform produces multiple sequences of approximation and detail coefficients at different scales, whereas a DCT transform produces a flat sequence of coefficients. The inverse transformation function T^{-1} converts given signal coefficients in Ψ into the signal f. The transform T is invertible, i.e. $T^{-1}(T(f)) = f$.

Thresholding function. The thresholding function is parametrized by the number of representation coefficients that are to be retained. The function retains the given number of coefficients *biggest in magnitude*; all other coefficients are set to zero.

Similarity metric. Having an original signal and a signal reconstructed from a compressed coefficient sequence, a similarity metric S is used to obtain an objective statistical comparison of these two signals. See Section 5.2.2 for a list of similarity metrics used in the sparsity estimation procedure.

The sparsity estimation procedure can be summarized to the following steps.

1. Compute $g = T(f)$, coefficient vector representing f in the analyzed basis.
2. Estimate the number of significant coefficients r.
3. Compute $C(r, g)$, thresholded coefficient vector.
4. Compute $f_r = T^{-1}(C(r, g))$, reconstruction of f from its compressed form.
5. Compute $m = S(f_r, f)$, statistical similarity of original and recovered signal.

5.2.2 Metrics of compression performance

The problem of measuring signal compression quality is well known in many scientific areas and is being a subject of ongoing research e.g. in the field of image processing [68]. In this section, we describe a set of objective measures for evaluating signal compression quality. These measures provide statistical means of summarizing the amount of signal distortion introduced by a lossy compression. However, it is important to point out that the requirements on "quality" can significantly differ across the application domains, and it is difficult or not possible (depending on the level of generality) to develop a general quality criterion. For instance, it has been observed that universally employed statistical criterions, such as the mean squared error and signal-to-noise ratio (see below), provide results that are often in contrast to subjective perception of a compressed image [69]. It is reasonable to expect such inconsistency in the area of CPU performance signals; therefore, the following metrics are only intended to provide comparisons between different compression methods and to provide a fundamental overview of the compression performance. The introduced metrics are most likely not sufficient for determining the suitability of a compression technique for specific application purposes.

The basic quantitative characterization of the reconstruction error is to compute its magnitude (norm):

Definition 19 (Polynomial norm of the reconstruction error)**.** Let $f, f' \in \mathbb{R}^n$ be the source and recovered signal, respectively. For a real $p \geq 1$, the l_p norm of the reconstruction error is

$$\|f - f'\|_{l_p} = (|f_1 - f'_1|^p + |f_2 - f'_2|^p + ... + |f_n - f'_n|^p)^{\frac{1}{p}}$$

In particular, the l_2 norm of the reconstruction error is

$$\|f - f'\|_{l_2} = \sqrt{\sum_i (f_i - f'_i)^2}$$

In signal processing, a common metric is the signal-to-noise ratio (SNR):

Definition 20 (Signal-to-noise ratio, [61])**.** Let $f \in \mathbb{R}^n$ be a signal. Then

$$SNR(f) = 10 \log_{10} \frac{\sigma_f^2}{D_f}$$

where σ_f^2 is the variance of f and D_f is distortion (or noise) present in the signal.

For the purposes of measuring reconstruction quality, the traditional definition of signal-to-noise ratio is adopted using the l_2 norm:

Definition 21 (Reconstruction signal-to-noise ratio, [52])**.** Let $f, f' \in \mathbb{R}^n$ be the source and recovered signal, respectively. The reconstruction SNR is

$$SNR(f, f') = 10 \log_{10} \frac{\|f\|}{\|f - f'\|}$$

In order to measure the energy of discarded coefficients in a signal spectrum, we use the notion of retained energy:

Definition 22 (Retained energy)**.** Let f_c, $\hat{f}_c$ be the original and thresholded coefficient vector in $\mathbb{R}^n$, respectively. The retained energy of the thresholded coefficient vector with respect to the original vector is

$$RE(f, \hat{f}_c) = \frac{\|\hat{f}_c\|_2}{\|f_c\|_2}$$

5.2.3 Estimating the compressibility

In lossy compression techniques, the compression ratio is a trade-off between reduction of the space complexity and information loss. A practical choice of the compression ratio highly depends on the particular application domain and requirements. Usually, experimentation with real data sets is required. In this section, we describe simple strategies that can be used to find an initial guess for the compression ratio.

Balancing sparsity and retained energy (BSRE)

The sparsity/energy balancing strategy finds the compression ratio r such that

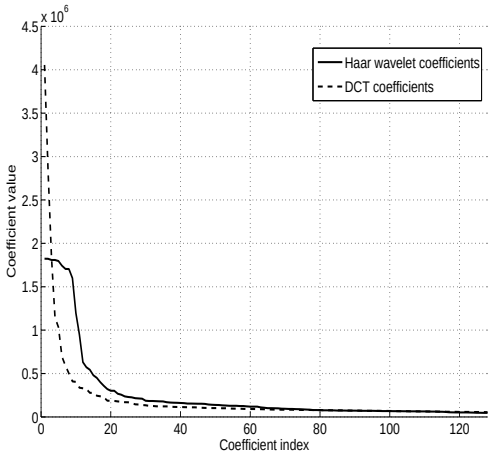
$$r = RE(T(f), C(r, T(f))$$

or in a more relaxed version, r is defined as

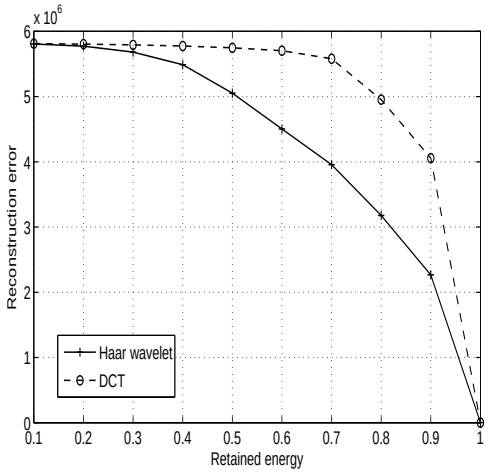
$$\min_r |r - RE(T(f), C(r, T(f))|$$

The idea behind this algorithm is that by increasing the compression ratio (i.e., the amount of discarded coefficients), we cause the $RE(T(f), C(r, T(f)) = \frac{\|C(r,T(f))\|}{\|T(f)\|}$ to decrease. Indeed, we discard the coefficients by setting them to zero, causing the fraction nominator to have a lower l_2 length. At some point, the compression ratio is the same as the retained energy of the coefficient vector; increasing the compression ratio lowers the information content and increasing the information content lowers the compression ratio.

In practice, the compression ratio and information change are not continuous, as the coefficient vector is finite. Therefore we get along by minimizing the difference $|r - RE(T(f), C(r, T(f))|$.



(a) Coefficient magnitudes of a performance signal



(b) Retained energy vs. error in signal domain

Figure 5.3: Both figures are based on the performance signal "Number of retired mispredicted branch instructions". The signal has length 1024 and was sampled while running the SPEC CPU2006 403.gcc benchmark. Figure (a) depicts coefficient magnitudes in wavelet and DCT representations; for the sake of visual clarity, the coefficients were linearized and ordered by magnitude of their absolute value. Only the first 120 coefficients in this ordering are shown. Figure (b) illustrates dependency between retained energy and reconstruction error. For each level of retained energy l, the coefficients of c were added to $\hat{c}$ in order of their importance stopping when retained energy l was achieved. Then the signal $\hat{f}$ was reconstructed from $\hat{c}$ and the reconstruction error was measured using l_2 metric.

Discarding non-significant coefficients (DNC)

The DNC strategy finds the compression ratio r such that

$$RE(T(f), C(r, T(f)) = K$$

or in a more relaxed version, r is found as

$$\min_r |K - RE(T(f), C(r, T(f))|$$

where K is a retained-energy constant, $0 < K < 1$.

Unlike the BSRE strategy where the compression ratio and the retained energy of the compressed coefficient vector are balanced, the DNC strategy applies a fixed constraint on the retained energy of the compressed coefficient vector and finds a corresponding compression ratio. For instance, by setting $K := 0.1$, the method finds a compression ratio such that only 10% falloff in $RE(T(f), C(r, T(f))$ is caused by applying the compression.

By analogy to BSRE, the definition using a minimum is proposed due to practical reasons.

5.2.4 Compressibility in the DCT basis

Figure 5.4 shows the number of significant coefficients detected by two different procedures (BSRE and DNC, Section 5.2.3) for the signals of the *403.gcc* benchmark. The estimates differ significantly across the space of evaluated signals. For instance, the number of executed multiply operations was detected to have 436, resp. 250 significant coefficients out of 1024. On the other hand, the frequency spectrum of the number of mispredicted branch instructions when running the same benchmark was highly concentrated into a small set of high-valued coefficients, so that the estimation procedures reported only 10, resp. 34 significant coefficients. This behaviour can be attributed to the properties of the DCT basis. As the basis is designed to represent periodic signals, it does extremely well for signals like the number of

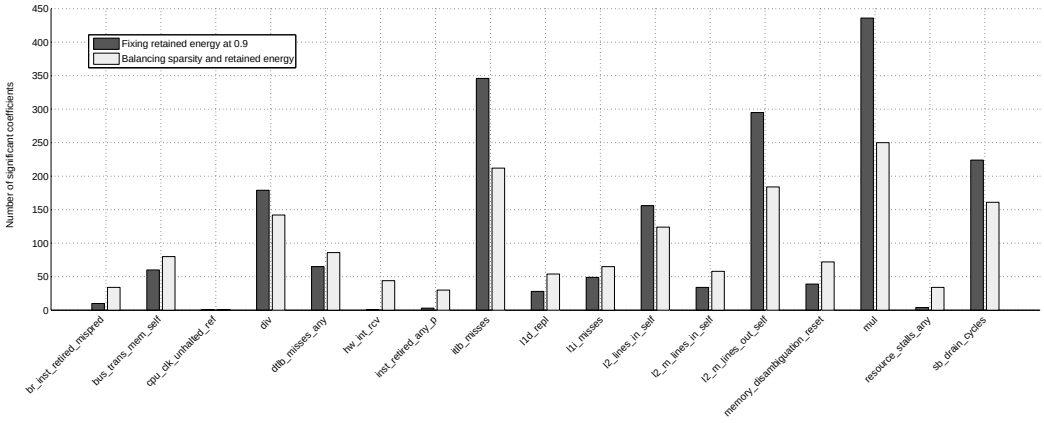


Figure 5.4: Compressibility of the *403.gcc* signals in the DCT basis. The signals from Table 5.2 were captured while running the SPEC CPU2006 403.gcc benchmark. The signal length is 1024.

reference clock ticks and the number of retired instructions, but it fails to concisely represent signals with sharp peaks (such as arithmetic operations in an integer-based benchmark).

Also note that for the DCT basis, fixing the retained energy at 90% yielded more pessimistic estimates than balancing the retained energy to the number of significant coefficients. This was because the *403.gcc* signals were generally not very sparse in the DCT basis and thus the BSRE estimation procedure tended to offset the higher retained energy (which was hard to achieve) against the number of significant coefficients.

5.2.5 Compressibility in selected wavelet bases

We evaluated compressibility in the following wavelet families which facilitate fast transforms:

Wavelet family	Description
Haar	A discrete, simplest possible wavelet.
Daubechies	Basic orthogonal compactly supported wavelet.
Coiflets	Orthogonal compactly supported wavelet with vanishing moments equally distributed for the scaling function and the wavelet.
Symlets	Orthogonal wavelet with maximum symmetry and compact support.
Biorthogonal wavelets	Wavelet family with different wavelets for decomposition and reconstruction.

Table 5.3: Selected wavelet families and their characteristics

Figure 5.5 summarizes the compressibility of the evaluated signals in five different wavelet bases. For each signal, the number of significant coefficients was estimated using two different procedures (BSRE and DNC, Section 5.2.3). We see that for the case of the *403.gcc* benchmark, all signals had less than 140 significant coefficients out of 1024. The simplest Haar wavelet allowed for the best or close to the best compressibility across all evaluated signals, even when compared to the more sophisticated wavelets. Some evaluated signals were extremely sparse in the wavelet domain, e.g. the events that are related to the number of floating point operations in the *403.gcc* benchmark.

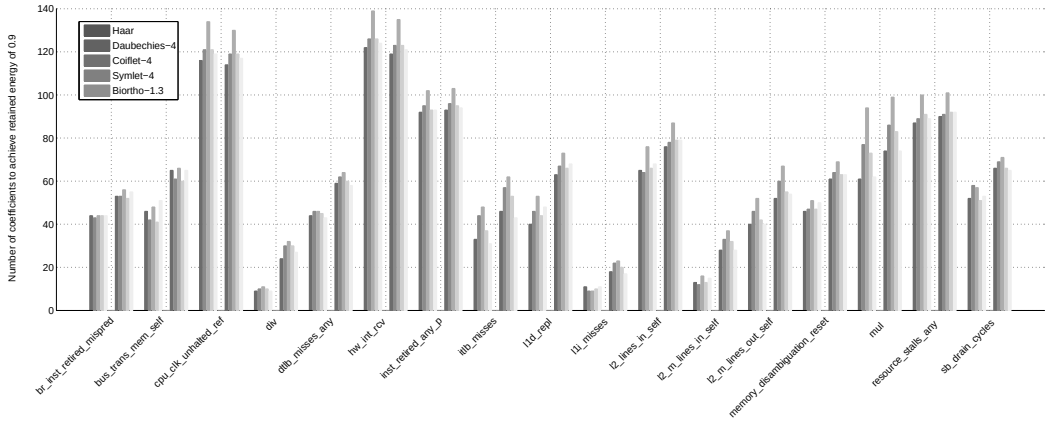


Figure 5.5: Compressibility of the *403.gcc* signals in selected wavelet bases. The signals from Table 5.2 were captured while running the SPEC CPU2006 403.gcc benchmark. The signal length is 1024.

5.2.6 Reconstruction quality

Removing coefficients in the sparsity domain can introduce errors in the signal domain after the reconstruction. The character of the error depends on the particular representation basis. Therefore, to get a complete information about the compressibility, we compare the reconstruction quality in different bases by using an objective measure. See Figure 5.6 which presents quality of the reconstruction when 128 significant coefficients out of 1024 in total were used. We see that for the *403.gcc* benchmark, the different wavelet bases again had approximately the same performance, and for the majority of the evaluated signals, the DCT basis had a worse performance than the evaluated wavelets.

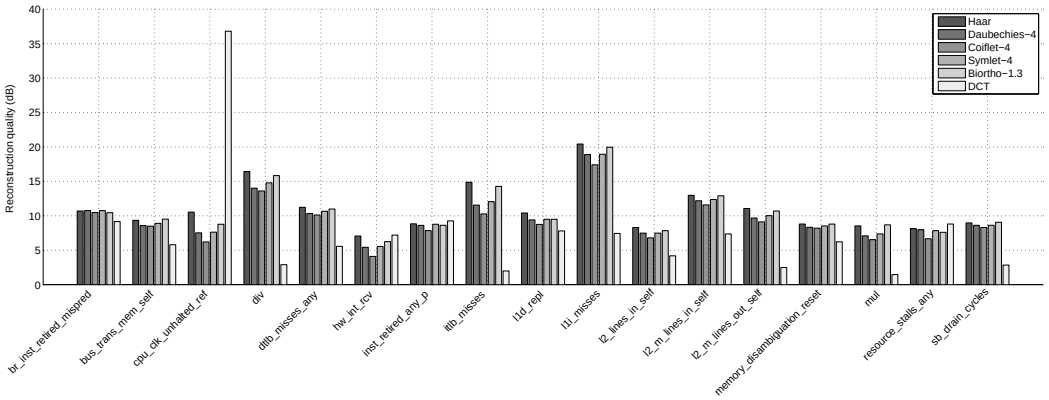


Figure 5.6: Reconstruction quality of the *403.gcc* signals. The signals from Table 5.2 were captured while running the SPEC CPU2006 403.gcc benchmark. The signal length is 1024.

5.2.7 Comparison of compressibility

Clearly, the compressibility estimates depend on the computational character of the workload. We have demonstrated the estimation for the *403.gcc* compiler, where the majority of signals were represented at best by the Haar wavelet. The more computational workloads, such as the fluid dynamics and raytracing, produced signals with repetitive patterns and high periodicity. For these workloads, the DCT basis gave the best sparsity estimates and reconstruction quality: for the *437.leslie3d* benchmark, the DCT basis provided a sparser representation than wavelets for 16 out of 17 evaluated signals. However, the Haar

wavelet showed stable performance across all benchmarks: the estimated sparsity was rarely higher than 20%.

It is interesting to note that the *relative sparsity* of the performance signal, computed with respect to the signal length, changes when computed for signals captured over different time periods. The longer the block size the more representative the block is of the signal and the greater the compressibility. Figure 5.7 illustrates this effect. The number of retired instructions was measured while running the *403.gcc* compiler. We divided the signal into blocks of sizes ranging from 64 bytes up to 16 Kbytes. For each block size, we determined how many coefficients are needed to retain 90% of energy in the wavelet domain.

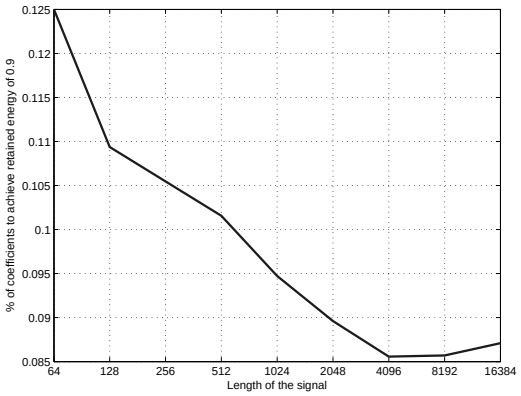


Figure 5.7: Compressibility of a 32kB signal (number of retired instructions) for different block sizes. For each block size, the signal was divided into chunks of respective length and the compressibility was measured. The figure depicts the median of the compressibility.

In all following experiments, the counter is read every 10 ms and the compressed signal transmitted after the counter has been read 128 times, i.e. approximately every second.

We conclude that a range of performance counters are approximately sparse when represented in the domains we looked at. Furthermore, the Haar wavelet performs as well as the more complicated wavelets. As it is computationally less demanding we shall restrict ourselves to consideration of the Haar wavelet and DCT in what follows.

Chapter 6

Compressive sampling of performance signals

In this chapter, we develop a systematic approach to evaluate performance of compressive sampling for the case of processor performance signals. First, we decompose the acquisition protocol into a set of modules. We describe the parametrization of the modules and design a toolkit for experimental evaluation. Then we use the toolkit to infer empirical properties of the most attractive implementation choices.

6.1 Modular view of compressive sampling

Compressive sampling (CS) can be viewed as a modular framework for signal compression. As such, it can be decomposed into a set of mutually interconnected algorithmic modules. Figure 6.1 captures the intrinsic structure of CS as it implicitly follows from the theorems in Section 3.3. The structure is adjusted to fit the task of compressing processor performance signals.

Let us define the parameter space of the CS framework to include the following quantities:

(i) Length of the sampling time slot

(ii) Type of the sampling basis

(iii) Sampling rate

(iv) Type of the representation basis

(v) Type of the reconstruction technique

This set of parameters influences different stages of compressive sampling acquisition protocol. We can further divide them into the parameters of the *encoding phase* (points (i), (ii), (iii)) and parameters of the *decoding phase* (points (iv), (v)).

6.2 Experimental implementation

To be able to discover the empirical properties of CS, we develop a simple experimental framework in the *Matlab* environment. We impose the following functional requirements on the implementation:

1. Blockwise processing of the signal
2. Interchangeable sampling matrices
3. Interchangeable representation matrices

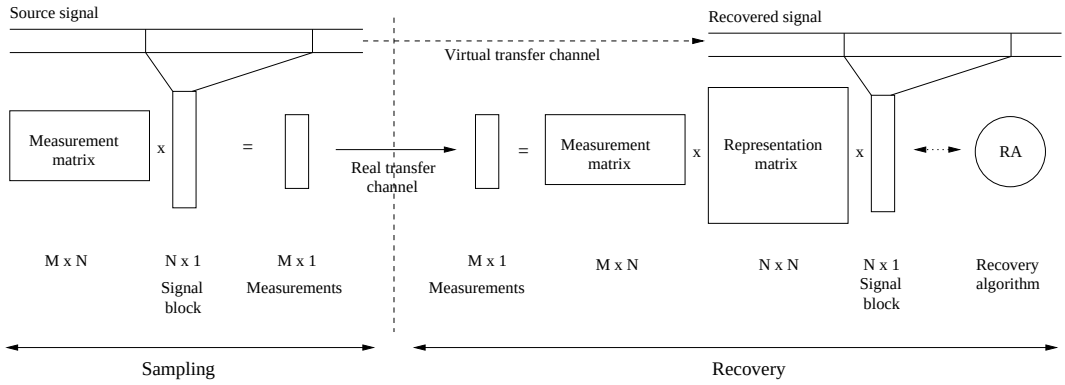


Figure 6.1: Compressive sampling parametrization

4. Interchangeable recovery algorithms
5. Automatic execution of the experiments and storage of the results
6. Extraction of the experimental results for purposes of data analysis

6.2.1 Sampling matrices

An instance of a sampling matrix is created by calling its constructor with syntax

```
h = matrixName(k, n)
```

The constructor creates a sampling matrix with dimensions $k \times n$, where k is the resulting number of measurements and n the length of the input signal block. The returned handle h points to a vector-by-matrix multiplication function of the form

```
y = computeMatrixMultiplication(mode, x)
```

where $mode = 1$ specifies multiplication of x by the sampling matrix and $mode = 2$ specifies multiplication of x by the transpose (pseudo-inverse) of the sampling matrix. Matrix-vector multiplication is the only allowed operation for the sampling matrix; it should always be sufficient for applications in compressive sampling. Moreover, if the sampling matrix implements the multiplication using a fast transform algorithm (such as the noiselet transform), multiplication by a vector is the only natural operation.

To give an example of a non-explicit sampling matrix, consider sampling in the noiselet domain (see Section 3.5). In this case, the sampling matrix first converts the signal block to the noiselet domain and then performs random downsampling of the coefficients. In other words, the signal block is transformed using only a random subset of the basis vectors. The key point is that the overall operation is presented as a matrix, but the noiselet transform is computed implicitly and also the random selection is implemented more efficiently (without a matrix multiplication).

We provide the following basic sampling matrices: random Gaussian sampling, random Bernoulli sampling, random noiselet sampling and random sampling in the time domain. For purposes of comparison, a regular downsampling matrix in the time domain is also provided. See Table 6.1 for a summary. Some matrices are provided in an orthogonalized version to ensure the rows are not linearly dependent and thus no redundant measurements are produced.

Function	Description
`randomGaussianMatrix(k, n)`	Explicit $k \times n$ matrix with entries drawn from the Gaussian distribution $N(0,1)$. Orthogonalized.
`randomBernoulliMatrix(k, n)`	Explicit $k \times n$ matrix with entries -1, $+1$ drawn from the symmetric Bernoulli distribution. Orthogonalized.
`randomGaussianMatrixNonOrthogonal(k, n)`	Explicit $k \times n$ matrix with entries drawn from the Gaussian distribution $N(0,1)$. Not orthogonalized.
`randomBernoulliMatrixNonOrthogonal(k, n)`	Explicit $k \times n$ matrix with entries -1, $+1$ drawn from the symmetric Bernoulli distribution. Not orthogonalized.
`randomNoiseletMatrix(k, n)`	Random downsampling in the noiselet domain.
`randomTimeDomainMatrix(k, n)`	Random downsampling in the time domain.
`regularTimeDomainMatrix(k, n)`	Regular downsampling in the time domain.

Table 6.1: Implemented sampling matrices.

6.2.2 Representation matrices

An instance of a representation matrix is created by calling its constructor with syntax

```
h = matrixName(n, ...)
```

The constructor creates a representation matrix with dimensions $n \times n$, where n is the length of the signal block. A representation matrix corresponds to a transformation between two n-dimensional bases, therefore it is always square. The additional parameters are representation specific, e.g. wavelets require a number of levels of the decomposition.

The returned handle h points to a matrix-vector multiplication function of the form

```
y = computeMatrixMultiplication(mode, x)
```

where $mode = 1$ specifies multiplication of x by the sampling matrix and $mode = 2$ specifies multiplication of x by the transpose (pseudo-inverse) of the sampling matrix. Multiplication by a given vector is the only allowed operation for a representation matrix; the reasoning is the same as in the case of sampling matrices.

Based on the experimental analysis of performance signals (see Chapter 5.2), we provide only the Haar wavelet and DCT representation matrix. Integration of other wavelet bases is straightforward. An identity matrix is also provided that is intended to be used with the interpolation recovery algorithms. See Table6.2. We use the *WaveLab* package [70] to compute the wavelet transformations.

Function	Description
`haarStandardRepresentationMatrix(n, l)`	Fast wavelet transformation of n length signal on l levels with the Haar filter. n must be a power of 2.
`dctRepresentationMatrix(n)`	Discrete cosine transformation of n length signal.
`identityRepresentationMatrix(n, l)`	Identity matrix with dimensions $n \times n$.

Table 6.2: Implemented representation matrices.

6.2.3 Recovery algorithms

In principle, the recovery algorithms are fully compatible with any combination of sampling and representation ensembles. The framework is provided with a set of optimization and greedy recovery algorithms, which are described in a greater detail in Chapter 3.4. For purposes of comparison, the set of recovery algorithms is augmented by interpolation mechanisms (linear interpolation and sinc interpolation,

[60]). The algorithms and their identification codes are summarized in Table 6.3. Implementation of the recovery algorithms is based on the *SparseLab* package [71].

Code	Description
`BP_EQ`	Basis pursuit with equality constraint.
`OMP`	Orthogonal matching pursuit.
`ROMP`	Regularized orthogonal matching pursuit.
`StOMP`	Stagewise orthogonal matching pursuit.
`CoSaMP`	Compressive sampling matching pursuit.
`LINEAR_INTERPOLATION`	Linear interpolation.
`SINC_INTERPOLATION`	Sinc interpolation.

Table 6.3: Implemented recovery algorithms.

The recovery algorithm works with a *recovery matrix* which is constructed internally (transparently to the framework user) as a combination of the sampling and representation matrix. We follow the convention of the *SparseLab* package and define the recovery matrix in the following way:

```
y = recoveryMatrix(mode, m, n, x, I, dim)
```

If $mode = 1$, the function computes multiplication of given vector x by the recovery matrix, if $mode = 2$, the function multiplies x by the transpose (pseudo-inverse) of the recovery matrix. The dimensions of the recovery matrix are $m \times dim$[1]. I is a subset of n columns of the matrix that are used to compute the product. Thus, by setting $I \subset \{1, ..., dim\}$ we multiply x by a recovery *submatrix* consisting only of columns from I. This functionality is important for greedy algorithms.

6.2.4 Blockwise processing of the signal

In our implementation of compressive sampling, the source signal is processed in blocks of fixed length. The blocks are encoded, transferred and decoded separately. At the end of the processing pipeline, the signal estimate is assembled from the decoded blocks.

The block processing is facilitated by the functions

```
encodedBlock = encodeSignalBlock(signalBlock, samplingMatrix)
decodedBlock = decodeSignalBlock(encodedBlock, samplingMatrix,
                                 representationMatrix, recoveryAlgorithm)
```

The function `encodeSignalBlock` creates an encoded block by taking measurements using given sampling matrix. The encoded block is transferred to the recovery component, where `decodeSignalBlock` computes an estimate of the original signal using the same sampling matrix, given representation matrix and given recovery algorithm. Note the typical property of CS acquisition protocol: at the time of sampling, we do not require to know in what basis we recover the signal. On the other hand, the sampling matrix has to be known to both parties.

6.2.5 Automatic experimental evaluation

The set of sampling and representation matrices, recovery algorithms and functions for per-block signal encoding/decoding is integrated in a batch evaluation framework.

On the signal level, we define the function

```
evaluationResult = evaluateForSignal(signal, csParametrization)
```

Given the signal and parametrization of compressive sampling, the function decomposes the input signal into blocks, encodes the blocks, decodes the blocks and assembles the output signal estimate. The *csParametrization* structure contains the fields listed in Table 6.4.

[1]Keep in mind that the matrix is implicit, i.e. there is no real $m \times dim$ recovery matrix in the memory, instead the matrix is computed by calling the representation and sampling transformations.

Field name	Description
`blockSize`	Size of the signal block, 2^d where $d > 5$
`measurementSize`	Number of measurements per block
`samplingMatrix`	Instantiated sampling matrix
`representationMatrix`	Instantiated representation matrix
`recoveryAlgorithm`	One of the codes from Table 6.3

Table 6.4: Content of the csParametrization structure.

A typical initialization of the *csParametrization* structure looks as follows:

```
csParametrization.blockSize            = 128;
csParametrization.measurementSize      = 32;
csParametrization.samplingMatrix       = randomNoiseletMatrix(32, 128);
csParametrization.representationMatrix = haarStandardRepresentationMatrix(3);
csParametrization.recoveryAlgorithm    = 'BP_EQ';
```

Results of the evaluation are stored in the `evaluationResult` structure, whose fields are summarized in Table 6.5.

Field name	Description
`recoveredSignal`	Recovered signal (in the representation domain)
`perBlockEncodingTimes`	Encoding time for each signal block
`perBlockDecodingTimes`	Decoding time for each signal block

Table 6.5: Content of the evaluationResult structure.

Batch functionality for a specified signal set is implemented in the `evaluateForSignalSet` function:

```
function evaluateForSignalSet(outputDirectory, inputDirectory, ...
                              platformName, samplingPeriodFilter, ...
                              cpuNumberFilter, benchmarkNameFilter, ...
                              evaluatedSamplingMatrices, ...
                              evaluatedBlockSizes, ...
                              evaluatedSamplingRatios, ...
                              evaluatedRepresentationMatrices, ...
                              evaluatedRecoveryAlgorithms, ...
                              numberOfTrials ...
                             )
```

The function processes all signals stored in the `inputDirectory` with respect to given filters (`samplingPeriodFilter`, `cpuNumberFilter`, `benchmarkNameFilter`). For each signal, the parametrization of CS (block size, sampling ratio, sampling matrix, representation matrix, recovery algorithm) is systematically set to all possible combinations of the evaluated values. For each such combination, the results are computed by calling `evaluateForSignal` repeatedly according to given `numberOfTrials`.

The evaluation results (Table 6.5) are stored into given `outputDirectory` together with information about the source signal. Each trial is stored in a separate file.

6.2.6 Extraction of results

A systematic traversal of the CS parameter space which is obtained by using the functions from Section 6.2.5 needs to be further processed in order to extract useful information from it. Typically, during a regression analysis one fixes a given set of parameters while a chosen parameter is observed as an independent variable. To facilitate this kind of analysis, a simple data extractor is provided.

First, the experimental data is loaded using a simple call to

```
results = loadResults(inputDirectory)
```

The results of the `evaluateForSignalSet` function stored in the `inputDirectory` are post-processed to contain a set of performance metrics for assessing the quality of reconstruction (e.g. signal-to-noise ratio, refer to Section 5.2.2 for an overview).

As a second step, a set of constraints on both the evaluation parameters and post-processed values is created using the functions

```
updateConstraintSet = addConstraint(constraintSet, variableName, variableValue)
updateConstraintSet = replaceConstraint(constraintSet, variableName, variableValue)
```

The set of constraints is built incrementally, in each step a new constraint is introduced. A constraint means that a variable with a given `variableName` is required to have a given `variableValue`. A variable is technically any field of the result structure, either present originally or generated during the post-processing step. For example, a constraint set can be built in the following way:

```
c = addConstraint([], 'measurementPeriod', '10ms');
c = addConstraint(c, 'csParametrization.blockSize', 128);
c = addConstraint(c, 'csParametrization.representationMatrixName', 'HAAR_STD');
c = addConstraint(c, 'csParametrization.recoveryAlgorithm', 'BP_EQ');
c = addConstraint(c, 'csParametrization.samplingMatrixName', 'NOISELETS');
```

In the final step, the set of constraints is applied to filter the results and extract a given independent variable:

```
[x, y] = extractResults(results, constraintSet, independentVariable, 'snr');
```

Here, `x` contains the set of distinct values of `independentVariable`. For each each value of `x`, `y` contains an *array* of values of `independentVariable`.

6.3 Experimental analysis

6.3.1 Measurement matrices

We performed a series of experiments to clarify empirical properties of selected random measurement bases. First, we considered whether there was a significant difference in the reconstruction quality between a signal sampled with the Bernoulli and Gaussian sampling matrices. These matrices work with high probability like an approximately orthogonal system. However, a concrete instance can contain linearly dependent rows producing redundant measurements. Therefore we further distinguish between measurements with (1) non-orthogonalized matrices (i.e. with entries directly sampled from the respective distribution) and (2) matrices with rows orthogonalized by the Gram-Schmidt procedure.

We fixed the compression ratio and took measurements by using orthogonalized Gaussian and Bernoulli matrices. For each matrix type and each signal, we performed 20 trials. In each trial, the measurement matrix was independently constructed and orthogonalized, the signal was measured and its estimate was computed. Figure 6.2 presents results for a set of *403.gcc* performance events. The results were comparable for all evaluated workloads. We conclude that the difference between using orthogonalized Gaussian and Bernoulli random matrices for sampling is not significant in terms of reconstruction quality.

In a practical implementation, the orthogonalization of matrices may not be readily achievable, e.g. when the matrix is implemented in hardware. Subsequently, sampling with non-orthogonalized matrices may be of interest. The Bernoulli matrix is a promising candidate for efficient hardware implementation, due to the fact that it requires only values of 1 and -1. Sampling with non-orthogonalized Bernoulli matrix was compared to sampling with orthogonalized Bernoulli matrix. Figure 6.3 shows the results for different sampling ratios and a fixed signal. It can be seen that the orthogonalized matrix in general outperformed the non-orthogonalized one. Orthogonalization was critically important

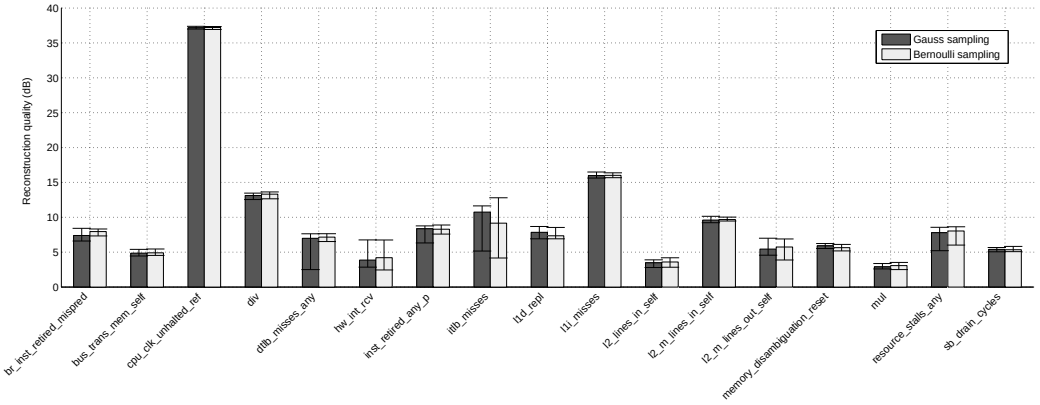


Figure 6.2: Comparison of reconstruction quality for sampling with random Bernoulli and Gaussian matrices. The events from Table 5.2 were sampled while running the SPEC CPU2006 403.gcc benchmark. The length of each signal is 1024 samples. The signals were recovered using OMP. Each reconstruction task was repeated in 20 independent trials. The bars depict median of reconstruction quality and the additional interval signs show range between the first and the third quartile.

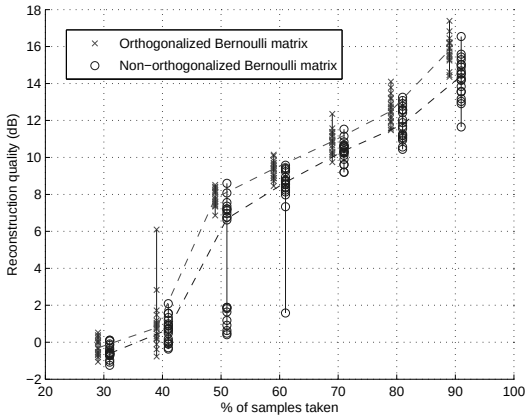


Figure 6.3: Comparison of reconstruction quality for orthogonalized and non-orthogonalized Bernoulli sampling matrix. For each compression ratio, we perform 20 independent trials.

in a certain region. The region corresponds to the boundary below which CS was no longer reliably performant.

Consider now the comparison of orthogonalized random Bernoulli matrix to random sampling in the noiselet domain. Noiselet measurements are complex numbers and, in general, require both the real and imaginary part of the measurements to be transferred. Therefore a fair comparison needs to be performed at a normalized sampling ratio where the number of real coefficients is the same. Figure 6.4 compares noiselets at a non-normalized and normalized sampling ratio (50%, resp. 25%) to a random Bernoulli matrix. When the sample rates were not normalized, noiselets achieved generally better reconstruction quality than Bernoulli matrices. This was expected due to their perfect incoherence to the Haar wavelet domain. However, after normalization noiselets achieved comparable or slightly worse reconstruction quality than the Bernoulli matrix.

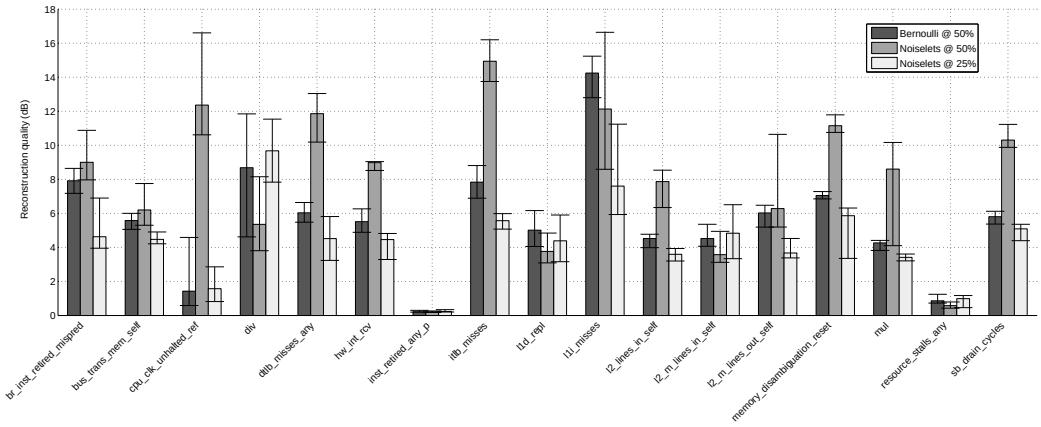
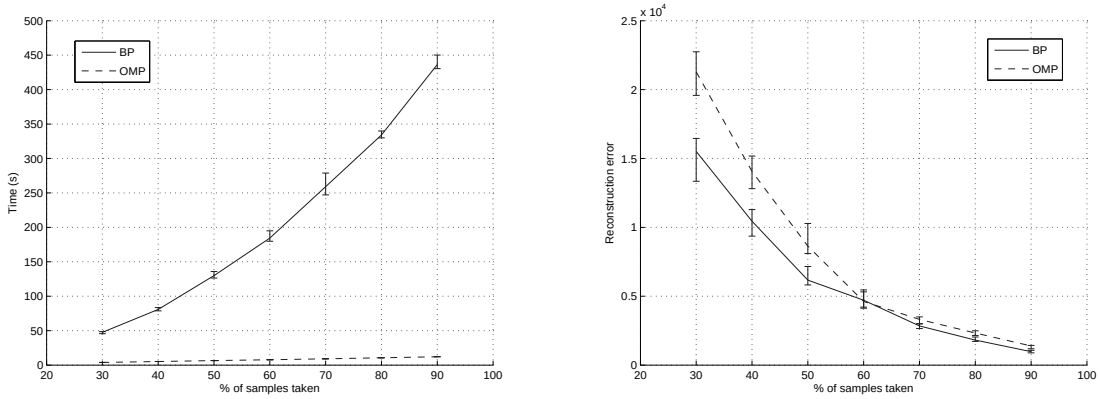


Figure 6.4: Comparison of reconstruction quality for the Bernoulli and noiselet sampling. For each evaluated signal and each sampling ratio, we performed 20 independent trials consisting of sampling and recovery. The signals from Table 5.2 were captured while running the SPEC CPU2006 403.gcc benchmark. The signal length is 1024. The bars depict median of reconstruction quality and the additional interval signs show range between the first and the third quartile. The recovery task was formulated as Basis Pursuit.



(a) Running time of BP vs. OMP

(b) Reconstruction error of BP vs. OMP

Figure 6.5: Comparison of BP and OMP for a fixed signal of length 1024. The signal represents the number of retired mispredicted branch instructions and was captured while running the SPEC CPU2006 403.gcc benchmark. The sampling and recovery was performed in 20 independent trials. Both algorithms recover the signal from random Gaussian projections. The plot depicts median of time/error and the additional interval signs show range between the first and the third quartile.

6.3.2 Recovery algorithms

Our measurements confirm the time bounds predicted by theory. For all tested signals and all reasonable sampling ratios, the greedy algorithms considerably outperformed optimization algorithms in processing time. The difference was typically orders of magnitude. Rather than presenting the results for all signals, we illustrate here the phenomenon in finer granularity for a fixed signal. Figure 6.5(a) depicts the running time of the BP and the OMP algorithms for different sampling ratios.

On the other hand, BP allowed more accurate recovery than OMP for many signals evaluated in this work. Figure 6.5(b) illustrates the quality differences for measurements with the Gaussian matrix.

6.3.3 Comparison to regular sampling

The evaluation compares the recovered signal using CS to the recovered signal using regular sampling followed by *sinc* interpolation [2] [72]. The regular sampling algorithm, for a measurement size M and a signal block of length N, takes M measurements in regular intervals of length $\frac{N}{M}$ from the signal block. The unknown values in the signal estimate (exactly $N - M$ values) are computed by sinc interpolation. Marginal entries (i.e. the first and the last block of unknown entries) are extrapolated. For clarity, we chose N and M such that $\frac{N}{M}$ is an integer.

There are multiple ways in which compressive sampling can be implemented. Based on our observations in the previous sections, we select three configurations with different qualitative properties and computing costs. The first two configurations aim at achieving an accurate reconstruction when information about the signal is known a priori. The *CS(Noiselets, Haar)* configuration takes samples in the noiselet domain and recovers the signal in the Haar domain. The *CS(Time, DCT)* configuration takes samples in the time domain and recovers the signal in the DCT domain. Both *CS(Noiselets, Haar)* and *CS(Time, DCT)* formulate the recovery task as Basis Pursuit and employ highly incoherent basis pairs (incoherency 1 and approximately $\sqrt{2}$, resp.). The third configuration targets a practical setting where the signals are sampled in the processor and recovered in real-time. This configuration uses the universally incoherent Bernoulli sampling and recovers the signal using the OMP algorithm, which was in our experiments empirically faster than BP. The sampling matrix is not orthogonalized and the signals are recovered in the Haar domain. We denote this configuration *CS(N-Bernoulli, Haar)*.

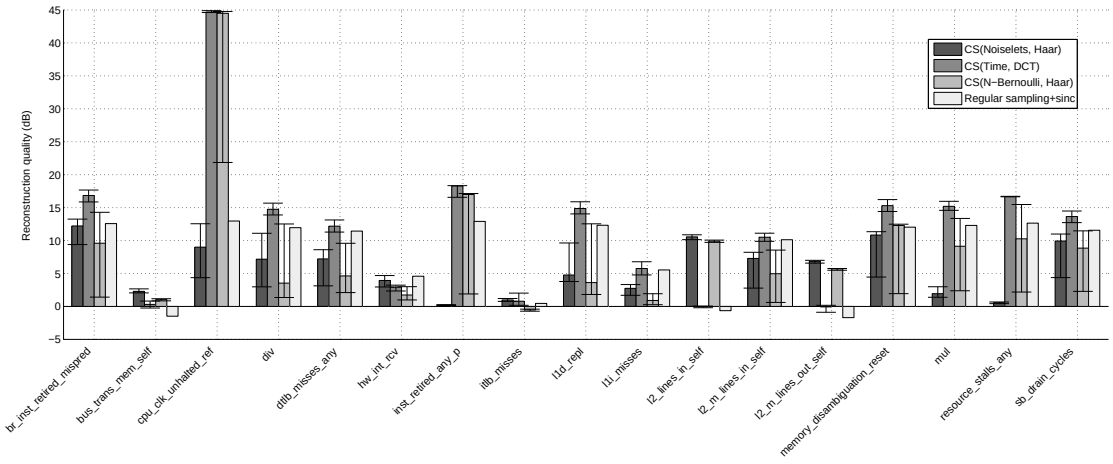


Figure 6.6: Comparison of compressive sampling to regular sampling followed by sinc interpolation. The signals from Table 5.2 were captured while running the SPEC CPU2006 453.povray benchmark. The signal length is 1024 and the sampling ratio is 50%. The bars depict median of reconstruction quality and the additional interval signs show the range between the first and the third quartile.

The evaluation was carried out for different sampling ratios. Figure 6.6 shows the results when 50% of samples were taken while running the *453.povray* benchmark. The majority of the signals were recovered at best by compressive sampling. 10 out of 13 signals which were highly sparse in the DCT domain were recovered in the best quality by *CS(Time, DCT)*. Two signals related to the L2-cache, highly sparse in the Haar wavelet domain, were recovered by *CS(Noiselets, Haar)* with $10dB$, resp. $5dB$ accuracy improvement over the other configurations.

The choice of the sparsity basis was less clear for the *403.gcc* benchmark and employing CS did not bring so unequivocal performance gain. *CS(Noiselets, Haar)* configuration recovered 4 signals better than regular sampling, namely the signals from the arithmetical units, TLB and L2 cache. These signals were highly sparse in the Haar wavelet domain. The efficient *CS(N-Bernoulli, Haar)* configuration recovered

[2] Given a sampling period T and uniformly spaced samples $f(nT)$, the approximation of f at point t is $f(t) = f(nT)\frac{sin(\pi(t-nT)/T)}{\pi(t-nT)/T}$

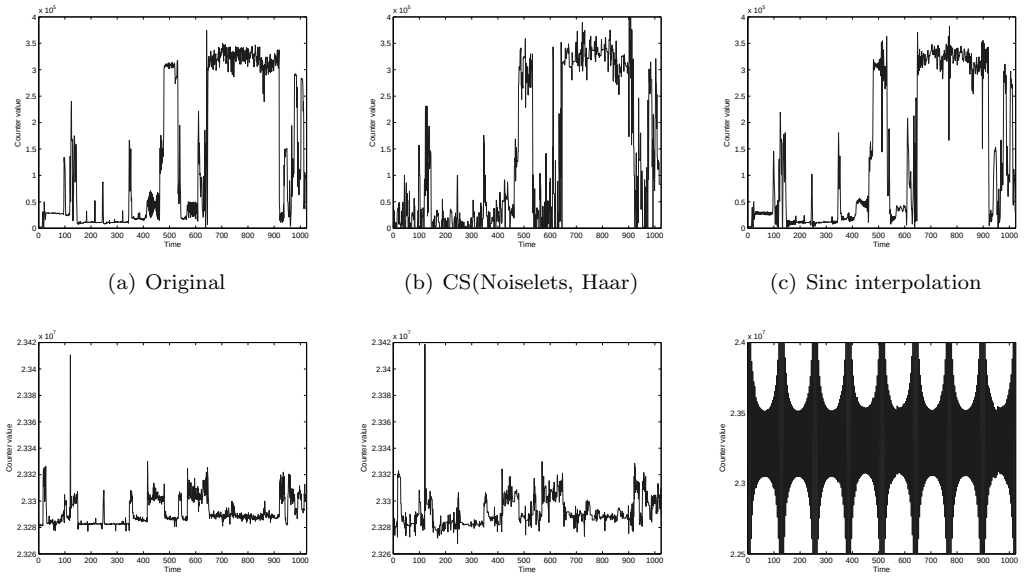
5 signals better and the *CS(Time, DCT)* only 1 signal, the number of processor cycles. Similar results were obtained for the *464.h264ref* benchmark.

In the fluid dynamics benchmark *437.leslie3d*, the reconstruction quality of CS was in many cases comparable to that of regular sampling. However, only 5 signals were recovered better by CS. When the sampling ratio was increased to 80% to compensate for the non-ideal incoherence of the time-DCT pair, *CS(Time, DCT)* was better in 15 out of 17 cases.

Higher compression ratios, e.g. taking 25% of samples, worked only for highly sparse signals, in this case for signals with sparsity estimates below 5% (see Figure 5.5). This corresponds to the reported lower bounds of $3K$ - $5K$ samples in [73]. For example, in the *403.gcc* benchmark, the number of multiply operations and the number of ITLB misses, both highly sparse in the Haar wavelet basis, were recovered the best by *CS(Noiselets, Haar)*.

In general, the evaluated signals which were recovered better by sinc inteporlation shared a strong periodic character. Because such signals are often sparse in the DCT basis, a comparable or better performance of CS was achievable once the sampling ratio was increased to compensate for the sparsity and incoherence.

CS also worked for the evaluated signals where interpolation resulted in a recovery which bore no resemblance to the original signal. Typically, these signals exhibit a non-smooth development with lots of discontinuities. Figure 6.7 shows an example of two contrasting signals. Both signals are captured by CS, but in one case CS is outperformed by regular sampling followed by sinc interpolation and in the other case CS is better.



(a) Original (b) CS(Noiselets, Haar) (c) Sinc interpolation

(d) Original (e) CS(Noiselets, Haar) (f) Sinc interpolation

Figure 6.7: Recovery of contrasting signals. Signal (a) represents the number of mispredicted branch instructions during the first 10s of SPEC 403.gcc compiler. Signal (d) represents the number of reference cycles from the same benchmark. The signal estimates (b), (c), (e), (f) were recovered from 50% of samples. Negative values in the estimates were set to 0. For the sake of legibility, signal (f) has a coarser scaling of the y-axis than (e).

6.3.4 Discussion

We have shown that for a range of different performance signals, the simple Haar wavelet or DCT can work surprisingly well. However, the sparsity and compression efficiency may be further enhanced by using more advanced representation techniques. Current research explores redundant and mixed representation dictionaries which allow for combining the advantages of different representation bases.

For instance, see [74] for results on using recovery dictionaries combining spikes and cosine waveforms or [1] for a demonstration of recovery with a highly overcomplete dictionary of Gabor functions.

The recovered estimates were often noisy; for basis pursuit, this has been observed even for simple artificial signals [73] and is attributed to a "leakage" of signal energy to the high frequency coefficients in the representational domain. See for example [73] for suggestions how the "leakage" can partially be reduced if signal-specific constraints in the recovery are imposed. To reduce the noise in greedy algorithms, it may be helpful to exploit structure of the representational domain (e.g. the Haar wavelet) and employ tree-based modifications of matching pursuit, see for instance [75, 76].

Naturally, the measure used for assessing reconstruction quality heavily depends on the evaluation objectives. We chose to measure performance by the objective l_2 metric and signal-to-noise ratio measures. A more specific analysis with a focus on prediction of the thread behaviour and new scheduling algorithms may require more tailored measures based on stochastic modeling and information theory.

In this work, CS was not run at its best possible configuration as predicted by theory. To be able to translate Basis Pursuit as a linear programming task, we chose the real-valued DCT basis instead of the complex-valued Fourier basis as the domain for signal recovery. Although DCT is still highly incoherent to the time domain, the Fourier basis would allow a perfect incoherence. Noiselets were evaluated at a normalized sampling rate which significantly affected their performance. The normalization was required because of the intrinsic usage of complex numbers in noiselets. We are currently designing a real sampling basis which is to be perfectly incoherent to the Haar wavelet and facilitates a fast transform algorithm. Having such a basis would bring substantial improvements in the reconstruction quality of CS.

We have observed strong inter-signal correlations between the evaluated performance signals. Such dependencies could allow for modeling the signal ensembles as jointly-sparse [77] and employing the algorithms for *distributed compressive sampling*. This approach could further lower the required bandwidth.

It must not be forgotten that the signals we used were themselves regularly sampled valued of the underlying performance counters. It is conceivable that a more accurate representation could be obtained by CS by directly working with the underlying signal in hardware.

Chapter 7

Per-core sampling module

In this chapter, we outline a proposal of a compressive sampling module intended to be used on per-chip or per-core basis [1]. The module is designed to perform the sampling part of compressive sampling (see Chapter 3), taking a real-valued signal of length n and producing its encoded form of length k, $k < n$. The signal is supposed to be sampled periodically in the time domain and the task of the module is to correlate the signal to a given sampling matrix. In the following algorithms, we try to exploit the nature of how the signal is obtained (as a series of incoming values) and how it is encoded (using matrix multiplication).

More formally, consider signal $f \in \mathbb{R}^n$ and sampling matrix A with dimensions $k \times n$, $k < n$. Note that this is an "undersampling"' matrix producing vectors in $\mathbb{R}^k$. We need to obtain the vector

$$d = A \cdot f$$

By definition of matrix multiplication, the i-th element of d can be computed as

$$d_i = \sum_j A_{ij} f_j$$

i.e. by correlating the signal f to the i-th line of the sampling matrix. A straightforward computation of d by evaluating all its coefficients in the described way would require storing the vector f and traversing its coefficients multiple times. In our case it is better to take one f coefficient at a time, process it and throw it away. By processing we mean computing everything that takes this coefficient into account. If we initialize $d_i := 0$ for $1 < i <= K$ and store it somewhere, then computing $d_i := d_i + A_{ik} f_k$ for signal values f_k, $k = 1, ...N$ does the job. The basic algorithm looks this way:

Data: $f \in R^n$
Result: $d \in R^k$
$d_i := 0$ for $i = 1, ...k$;
foreach f_k *in* f **do**
  foreach d_i *in* d **do**
    $d_i := d_i + A_{ik} f_k$;
  end
end

Algorithm 4: Multiplying a signal by a matrix: a basic approach

Now we are concerned with the multiplication matrix A. For given fixed k, we need only one column of the A matrix to execute the inner *for* loop. After the k is advanced, the column is advanced as well and the old one is no more needed. Suppose we have a vector a that is a projection of the k-th column of matrix A. Then Algorithm 4 can be rewritten:

[1] The content of this chapter is subject of the U.S. Patent Application Nr. CH9-2008-022, METHOD AND APPARATUS FOR EFFICIENT GATHERING OF INFORMATION IN A MULTICORE SYSTEM, filed in June 2008 by the IBM Corporation.

Data: $f \in R^n$
Result: $d \in R^k$
$d_i := 0$ for $i = 1, ...k$;
foreach f_k *in* f **do**
  $a := A_{*k}$;
  foreach d_i *in* d **do**
    $d_i := d_i + a_i * f_k$;
  end
end

Algorithm 5: Multiplying a signal by a matrix: separation of the column

In Algorithm 5, A_{*k} denotes the k-th column of matrix A. Now we clearly see that the inner loop can be parallelized in terms of i - having f_k and a, d can be computed in one step provided the additions and multiplications run in parallel.

The described algorithm performs encoding of one signal block of length n. After that, the encoded vector d is sent away and the procedure runs the same way again. The important fact is that the sampling matrix A remains the same for all encoding blocks. Thus, the a vector will be consequently filled with the same values as new f_k samples arrive. From a's point of view, there is a cycle of fixed set of values for each a_i.

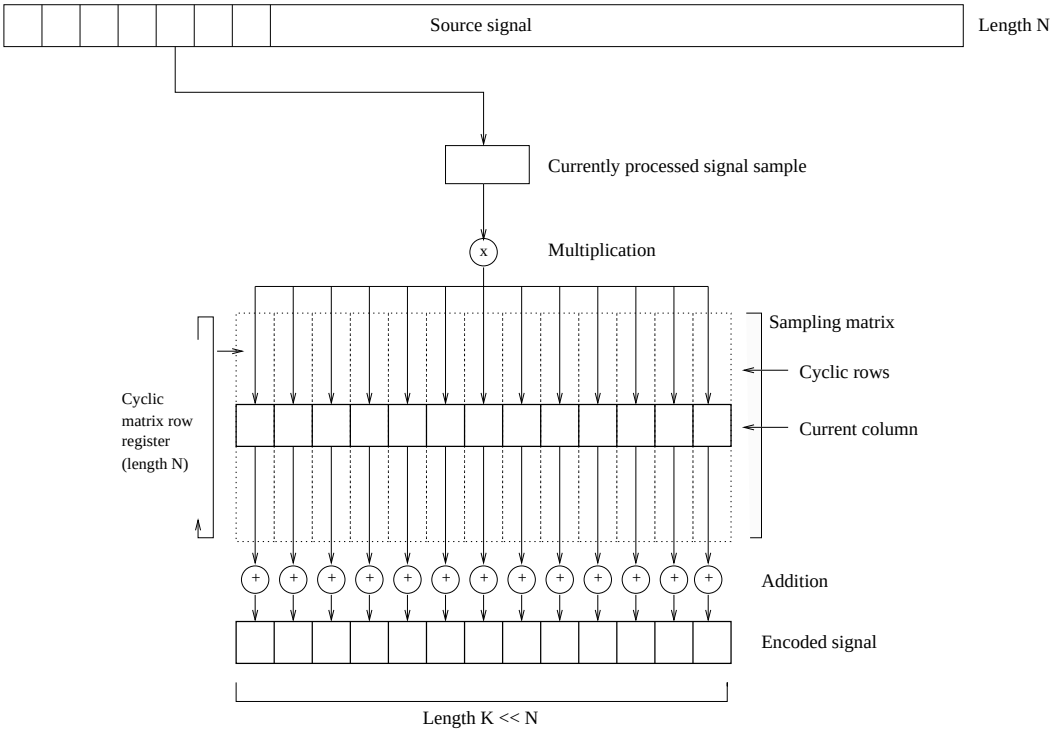


Figure 7.1: Schema of the per-core compressive sampling module.

It is possible to exploit special properties of the sampling matrices used in compressive sampling, so that the value cycle for a can be computed rather than stored. For example, columns of a random binary matrix with the symmetric Bernoulli distribution (see Theorem 5) can be generated using pseudo-random number generators of uniformly distributed integer variables. For instance, linear congruential generators [78, 79], lagged Fibonnaci generators [78] and shift-register generators [78] are the simple and well known ones. These algorithms are fairly easy to implement and implementations in both software and hardware exist (e.g. [80]). The fact that the algorithms behave deterministically upon initialization by a seed is important; the sampling matrix has to be known to the recovery algorithm, which is typically implemented in a different hardware of software component.

Chapter 8

Conclusion

This work is the first proposal and examination of dynamic compression of information gathered from performance monitoring units in multicore processors. For this task, conventional compression mechanisms are impractical: they require first the acquisition of the entire signal in a processor core, and then the application of a possibly computationally intensive compression algorithm. An alternative is desirable.

We evaluated the practicality of such an alternative, *compressive sampling* (CS). It is a surprising new development in information theory [81, 43], whose key result is that general classes of signals may be compressed non-adaptively, without acquiring the entire signal. These techniques require that a basis of representation for the signal be found such that the signal is (approximately) sparse (i.e. most of its coefficients are zero) when represented in that basis and that a second basis, the measurement basis, exists in which the signal representation is spread out over many coefficients. Moreover, it can suffice to choose a random basis as the measurement basis.

The mathematical foundation for compressive sampling was outlined. The mathematical theorems were categorized according to their assumptions and different approaches to the signal recovery were examined. The incoherence of noiselet and wavelet bases was proven using a new definition of noiselets based on the Kronecker product.

We gave an overview of contemporary processor architectures which exhibit a high degree of parallelism and performed a survey on how performance information can be gathered from them and exploited. As one of the most important applications of this performance data procurement, we considered the scheduling of threads on processor cores. Many proposals exist in the literature on performance aware schedulers whose main input for decisions is performance counters.

Our approach was motivated by demonstrating that many performance signals are compressible in the Haar wavelet and DCT representations. We chose a subset of well known benchmarks as the workload and analyzed properties of the signals gathered on a widely spread processor platform. We then evaluated distinct measurement and representational basis pairs, emphasizing measurement bases which can be practically implemented in hardware. We investigated the practicality of recovering the signal through an evaluation of the various algorithms proposed in the literature. Finally, having identified practical measurement and representational bases and efficient algorithms, the effectiveness of compressive sampling in compressing performance signals was measured. It was demonstrated that some of the signals can be accurately recovered from as low as a 25% sampling rate using compressive sampling, figures which correspond to those predicted by theory.

In general, we outlined a methodology which can be used for choosing, given a particular application, the most appropriate components of CS. A special purpose sampling algorithm was designed that efficiently compresses performance readings in a processor core.

CS proved an efficient alternative for some real-world, highly sparse signals. Further developments are needed in order to exploit its full potential for a broader range of signals, a reasonable expectation given CS is a new and active field. For example, performance could be improved by use of maximally incoherent measurement bases that consist solely of real numbers.

Bibliography

[1] E. J. Candès and M. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21 – 30, 2008.

[2] E. J. Candès, "Compressive sampling," in *Proceedings of the International Congress of Mathematicians*, vol. 3, pp. 1433 – 1452, 2006.

[3] J. Romberg and M. Wakin, "Compressed sensing: A tutorial." IEEE Statistical Signal Processing Workshop, 2007.

[4] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, "IBM POWER6 Microarchitecture," *IBM Journal of Research and Development*, vol. 51, no. 6, 2007.

[5] A. S. Tanenbaum, *Modern Operating Systems*. Prentice Hall, 2001.

[6] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia distributed monitoring system: Design, implementation and experience," *Parallel Computing*, vol. 30, no. 7, 2003.

[7] R. van der Pas, "Memory hierarchy in cache-based systems." Technical Report 817-0742-10, Sun Microsystems, 2002.

[8] M. S. Squillante and E. D. Lazowska, "Using processor-cache affinity information in shared-memory multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 131–143, 1993.

[9] C. D. Antonopoulos, D. S. Nikolopoulos, and T. S. Papatheodorou, "Realistic workload scheduling policies for taming the memory bandwidth bottleneck of SMPs," in *Proceedings of the International Conference on High Performance Computing*, pp. 286–296, 2004.

[10] Intel Corporation, *Intel 64 and IA-32 Architectures Software Developer's Manual*, vol. 3B. 2008.

[11] Intel Corporation, *Intel Itanium 2 Processor Reference Manual For Software Development and Optimization*. 2004.

[12] IBM Corporation, *Cell Broadband Engine Programming Handbook*. 2007.

[13] AMD Corporation, *BIOS and Kernel Developer's Guide for the AMD Athlon 64 and AMD Opteron Processors*. 2006.

[14] MIPS Technologies, *MIPS R10000 Microprocessor User's Manual*. 1996.

[15] S. Eranian, "Perfmon2: a flexible performance monitoring interface for Linux," in *Ottawa Linux Symposium 2006: Proceedings of the Linux Symposium*, vol. 1, pp. 269 – 288, 2006.

[16] M. Petterson, "Perfctr documentation." Available online from http://user.it.uu.se/ mikpe/linux/perfctr, 2008.

[17] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci, "A scalable cross-platform infrastructure for application performance tuning using hardware counters," in *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, (Washington, DC, USA), p. 42, IEEE Computer Society, 2000.

[18] S. Parekh, S. Eggers, and H. Levy, "Thread-sensitive scheduling for SMT processors." University of Washington Technical Report, 2000.

[19] J. Aas, "Understanding the linux 2.6.8.1 cpu scheduler." On–line article, http://joshaas.net/linux/linux_cpu_scheduler.pdf.

[20] A. Tucker and A. Gupta, "Process control and scheduling issues for multiprogrammed shared-memory multiprocessors," *SIGOPS Operating Systems Review*, vol. 23, no. 5, pp. 159–166, 1989.

[21] A. Fedorova, M. Seltzer, C. Small, and D. Nussbaum, "Performance of multithreaded chip multiprocessors and implications for operating system design," in *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, (Berkeley, CA, USA), pp. 26–26, USENIX Association, 2005.

[22] J. Nakajima and V. Pallipadi, "Enhancements for hyper-threading technology in the operating system: seeking the optimal scheduling," in *WIESS'02: Proceedings of the 2nd conference on Industrial Experiences with Systems Software*, (Berkeley, CA, USA), pp. 3–3, USENIX Association, 2002.

[23] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous multithreaded processor," in *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, (New York, NY, USA), pp. 234–244, ACM, 2000.

[24] C. Antonopoulos, D. Nikolopoulos, and T. Papttheodorou, "Scheduling algorithms with bus bandwidth considerations for SMPs," in *Proceeding of the 2003 International Conference on Parallel Processing*, pp. 547–554, 2003.

[25] J. Corbalán, X. Martorell, and J. Labarta, "Performance-driven processor allocation," in *OSDI'00: Proceedings of the 4th conference on Symposium on Operating System Design & Implementation*, (Berkeley, CA, USA), pp. 5–5, USENIX Association, 2000.

[26] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," in *ISCA '98: 25 years of the international symposia on Computer architecture (selected papers)*, (New York, NY, USA), pp. 533–544, ACM, 1998.

[27] A. Snavely, D. M. Tullsen, and G. Voelker, "Symbiotic jobscheduling with priorities for a simultaneous multithreading processor," in *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 66–76, ACM, 2002.

[28] R. L. McGregor, C. D. Antonopoulos, and D. S. Nikolopoulos, "Scheduling algorithms for effective thread pairing on hybrid multiprocessors," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, (Washington, DC, USA), p. 28.a, IEEE Computer Society, 2005.

[29] A. Fedorova, M. Seltzer, and M. D. Smith, "Cache-fair thread scheduling for multi-core processors," in *Technical Report TR-17-06, Harvard University*, 2006.

[30] M. D. Vuyst, R. Kumar, and D. Tullsen, "Exploiting unbalanced thread scheduling for energy and performance on a CMP of SMT processors," in *Proceedings of 20th International Parallel and Distributed Processing Symposium*, 2006.

[31] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," *SIGARCH Computer Architecture News*, vol. 32, no. 2, p. 64, 2004.

[32] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, (New York, NY, USA), pp. 135–140, ACM, 2001.

[33] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam, "vEC: virtual energy counters," in *PASTE '01: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, (New York, NY, USA), pp. 28–31, ACM, 2001.

[34] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, (Washington, DC, USA), p. 93, IEEE Computer Society, 2003.

[35] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos, "Online power-performance adaptation of multithreaded programs using hardware event-based prediction," in *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, (New York, NY, USA), pp. 157–166, ACM, 2006.

[36] A. Weissel and F. Bellosa, "Process cruise control: event-driven clock scaling for dynamic power management," in *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, (New York, NY, USA), pp. 238–246, ACM, 2002.

[37] M. Gomaa, M. D. Powell, and T. N. Vijaykumar, "Heat-and-run: leveraging SMT and CMP to manage power density through the operating system," *SIGARCH Computer Architecture News*, vol. 32, no. 5, pp. 260–270, 2004.

[38] R. Mooney, K. P. Schmidt, and R. S. Studham, "NWPerf: a system wide performance monitoring tool for large Linux clusters," in *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, (Washington, DC, USA), pp. 379–389, IEEE Computer Society, 2004.

[39] M. J. Sottile and R. G. Minnich, "Supermon: A high-speed cluster monitoring system," in *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, (Washington, DC, USA), p. 39, IEEE Computer Society, 2002.

[40] E. Anderson and D. Patterson, "Extensible, scalable monitoring for clusters of computers," in *LISA '97: Proceedings of the 11th USENIX conference on System administration*, (Berkeley, CA, USA), pp. 9–16, USENIX Association, 1997.

[41] R. Buyya, "PARMON: a portable and scalable monitoring system for clusters," *Software - Practice and Experience*, vol. 30, no. 7, pp. 723–739, 2000.

[42] D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, 2001.

[43] E. Candès and T. Tao, "Near optimal signal recovery from random projections and universal encoding strategies," *IEEE Transactions on Information Theory*, vol. 52, no. 12, pp. 5406 – 5425, 2006.

[44] E. Candès and T. Tao, "Decoding by linear programming," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4203 – 4215, 2005.

[45] E. Candès, J. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, no. 8, pp. 1207 – 1223, 2006.

[46] M. Rudelson and R. Vershinin, "Sparse reconstruction by convex relaxation: Fourier and gaussian measurements," in *Proceedings of the 40th Annual Conference on Information Sciences and Systems*, pp. 207 – 212, 2006.

[47] E. Candès and J. Romberg, "Sparsity and incoherence in compressive sampling," *Inverse Problems*, vol. 23, no. 3, pp. 969–985, 2007.

[48] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM Review*, vol. 43, no. 1, pp. 129–159, 2001.

[49] E. Candès and T. Tao, "The dantzig selector: statistical estimation when p is much larger than n," *Annals of Statistics*, vol. 35, no. 6, pp. 2313 – 2351, 2007.

[50] M. Figueiredo, R. Nowak, and S. Wright, "Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems," *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, no. 4, pp. 586 – 597, 2007.

[51] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3397 – 3415, 1993.

[52] D. Needell and J. A. Tropp, "CoSaMP: Iterative signal recovery from incomplete and inaccurate samples," *Preprint*, 2008.

[53] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," in *Proceedings of the 27 th Annual Asilomar Conference on Signals, Systems, and Computers*, pp. 40 – 44, 1993.

[54] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal mathching pursuit," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4655 – 4666, 2007.

[55] D. Needell and R. Vershynin, "Uniform uncertainity principle and signal recovery via regularized orthogonal matching pursuit," 2007. Submitted for publication.

[56] I. Daubechies, M. Defrise, and C. DeMol, "An iterative thresholding algorithm for linear inverse problems," *Communication on Pure and Applied Mathematics*, vol. 57, pp. 1413 – 1457, 2003.

[57] A. Gilbert, M. Strauss, J. Tropp, and R. Vershynin, "Algorithmic linear dimension reduction in the ell-1 norm for sparse vectors," 2006. Submitted for publication.

[58] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mapping into hilbert space," *Contemporary Mathematics*, vol. 26, pp. 189 — 206, 1984.

[59] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin, "One sketch for all: fast algorithms for compressed sensing," in *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, (New York, NY, USA), pp. 237–246, ACM, 2007.

[60] G. Strang, *Introduction to applied mathematics*. Wellesley - Cambridge Press, 1986.

[61] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding*. Prentice Hall, Authors, 2007.

[62] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[63] I. Daubechies, *Ten lectures on wavelets*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.

[64] R. Coifman, F. Geshwind, and Y. Meyer, "Noiselets," *Applied and Computational Harmonic Analysis*, vol. 10, no. 1, pp. 27 – 44, 2001.

[65] A. J. Laub, *Matrix Analysis for Scientists and Engineers*. SIAM, 2005.

[66] B. Falkowski and S. Rahadja, "Walsh-like functions and their relations," in *IEE Proceedings on Vision, Image and Signal Processing*, vol. 143, pp. 279 – 284, 1996.

[67] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[68] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error measurement to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[69] A. Eskicioglu and P. Fisher, "Image quality measures and their performance," *IEEE Transactions on Communications*, vol. 43, no. 12, pp. 2959 – 2965, 1995.

[70] J. Buckheit and D. Donoho, "Wavelab and reproducible research," in *Wavelets in Statistics*, pp. 55–81, Springer-Verlag, 1995.

[71] D. Donoho et al., "Sparselab: Seeking sparse solutions to linear systems of equations." Available online at http://sparselab.stanford.edu.

[72] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time signal processing.* Prentice Hall, 1999.

[73] E. J. Candès and J. Romberg, "Signal recovery from random projections," in *Proceedigns of SPIE Conference on Computational Imaging III*, pp. 76 – 86, 2004.

[74] H. Rauhut, K. Schass, and P. Vandergheynst, "Compressed sensing and redundant dictionaries," *IEEE Transactions on Information Theory*, vol. 54, pp. 2210–2219, May 2008.

[75] C. La and M. Do, "Signal reconstruction using sparse tree representation," in *Proceedings of SPIE on Wavelets XI*, vol. 5914, 2004.

[76] P. Jost, P. Vandergheynst, and P. Frossard, "Tree-based pursuit: Algorithm and properties," *IEEE Transactions on Signal Processing*, vol. 54, pp. 4685–4697, December 2006.

[77] M. F. Duarte, S. Sarvotham, D. Baron, M. B. Wakin, and R. G. Baraniuk, "Distributed compressed sensing of jointly sparse signals," in *Proceedings of the 2005 Asilomar Conference on Signals, Systems, and Computers*, pp. 1537 – 1541, 2005.

[78] G. Marsaglia, "A current view of random number generators," in *Computer Science and Statistics: 16th Proceedings of the Symposium on the Interface*, 1984.

[79] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Addison-Wesley, 1997.

[80] Y. Bi, G. D. Peterson, G. L. Warren, and R. J. Harrison, "Hardware acceleration of parallel lagged-fibonacci pseudo random number generation," in *Proceedings of the 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms*, pp. 215–218, 2006.

[81] D. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289 – 1306, 2006.

Appendix A

Examples of real performance signals

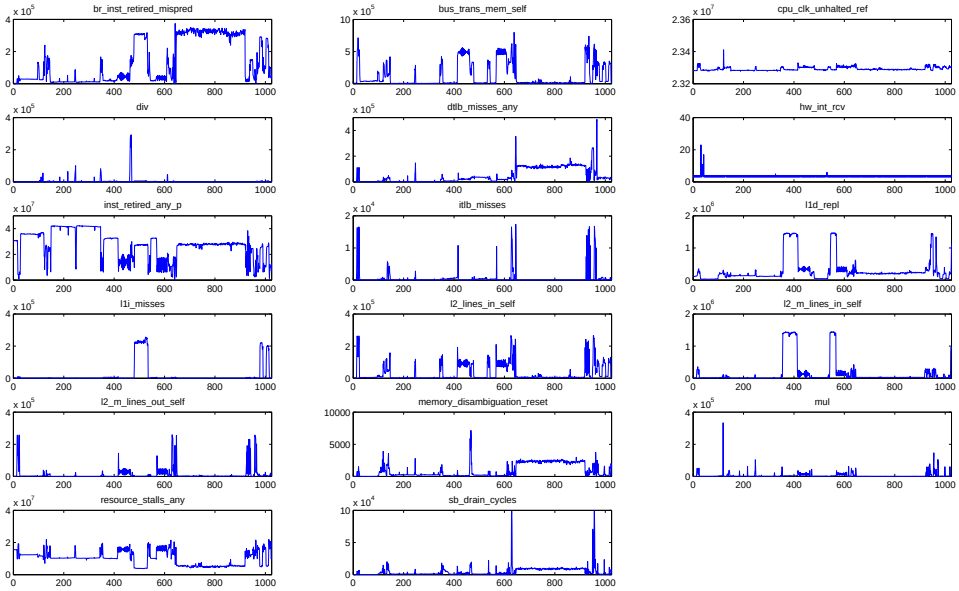


Figure A.1: Intel® Core 2 performance signals for SPEC CPU2006 403.gcc. Sampling frequency 10ms, 1024 samples.

Appendix B

Contents of the CD

The master thesis comes with a CD which contains the text of the work, images, traces and results of the experiments. The CD is structured as follows:

`/Text/Source/`	Text of the work in the LaTeXformat.
`/Text/Images/`	Images used in the text in the EPS format.
`/Text/Text.pdf`	Compiled version of the work as it was printed in the PDF format.
`/Data/Traces/`	The traces of evaluated performance counters in the *pfmon* ASCII format.
`/Data/Results/`	Results of the experiments, one directory per experiment.
`/Data/Results/readme.txt`	Configuration of the particular experiments.