Malleable Contextual Partitioning and Computational Dreaming

Gurkanwal Singh Brar

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

December 4, 2014
Blacksburg, VA

Keywords: Parallel Computing, Context Aware Computing, Computational Dreaming, Multiple Instruction Single Datastream (MISD) Computing, Brain Inspired Computer Architecture, Speech Recognition

Malleable Contextual Partitioning and Computational Dreaming

Gurkanwal Singh Brar

Abstract

Computer Architecture is entering an era where hundreds of Processing Elements (PE) can be integrated onto single chips even as decades-long, steady advances in instruction, thread level parallelism are coming to an end. And yet, conventional methods of parallelism fail to scale beyond 4-5 PE's, well short of the levels of parallelism found in the human brain. The human brain is able to maintain constant real time performance as cognitive complexity grows virtually unbounded through our lifetime. Our underlying thesis is that contextual categorization leading to simplified algorithmic processing is crucial to the brains performance efficiency. But, since the overheads of such reorganization are unaffordable in real time, we also observe the critical role of sleep and dreaming in the lives of all intelligent beings. Based on the importance of dream sleep in memory consolidation, we propose that it is also responsible for contextual reorganization. We target mobile device applications that can be personalized to the user, including speech, image and gesture recognition, as well as other kinds of personalized classification, which are arguably the foundation of intelligence. These algorithms rely on a knowledge database of symbols, where the database size determines the level of intelligence. Essential to achieving intelligence and a seamless user interface however is that real time performance be maintained. Observing this, we define our chief performance goal as: *Maintaining constant real time performance against ever increasing algorithmic and architectural complexities*. Our solution is a method for Malleable Contextual Partitioning (MCP) that enables closer personalization to user behavior. We conceptualize a novel architectural framework, the Dream Architecture for Lateral Intelligence (DALI) that demonstrates the MCP approach. The DALI implements a dream phase to execute MCP in ideal MISD parallelism and reorganize its architecture to enable contextually simplified real time operation. With speech recognition as an example application, we show that the DALI is successful in achieving the performance goal, as it maintains constant real time recognition, scaling almost ideally, with PE numbers up to 16 and vocabulary size up to 220 words.

Acknowledgements

I would like to thank Dr. JoAnn Paul whose support, constant encouragement and patience, was invaluable to this research. A special thanks to my parents, Dr. Gurvinder Singh and Mrs. Amritpal Kaur for instilling in me an appreciation for science and providing continued support towards my pursuits. Also, thanks to Mrs. Simran Brar Ghotge and Mr. Ved Ghotge for giving me a home away from home. Thanks to Dr. Cameron Patterson and Dr. Benjamin Knapp for serving on my committee. Thanks to my friends in India, Dubai, Dominican Republic and here in the USA for their support, with a special mention to Mr Gurpratap Thiara for inadvertently setting me on the path towards my Masters. Finally, I am indebted to the Guru Granth Sahib for keeping me in good stead through the years.

This research was supported in part by the National Science Foundation under grant 1043341. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

PE: Processing Element

MCP: Malleable Contextual Partitioning

DALI: Dream Architecture for Lateral Intelligence

DLP: Data Level Parallelism

TLP: Thread Level Parallelism

ILP: Instruction Level Parallelism

CLP: Context Level Parallelism

CP: Contextual Partitioning

MCP: Malleable Contextual Partitioning

CD: Computational Dreaming

NREM: Non Rapid Eye Movement

REM: Rapid Eye Movement

HMM: Hidden Markov Model

ANN: Artificial Neural Network

SISD: Single Instruction Single Data

MIMD: Multiple Instruction Multiple Data

SIMD: Single Instruction Multiple Data

MISD: Multiple Instruction Single Data

GPU: Graphics Processing Unit

GPGPU: General Purpose Graphics Processing Unit

SMT: Simultaneous Multithreading

SoC: System-on-Chip

CPU: Central Processing Unit

NoC: Network-on-Chip

CLP: Context Level Parallelism

CPM: Context Performance Metric

CAP: Context Activation Processor

CSN: Context Sensitive Network

HTK:	Hidden Markov Model Toolkit
CMU:	Carnegie Mellon University
PM:	Periodic Memory
TVS:	Threshold Vocabulary Size
MESH:	Modeling Environment for Software and Hardware
SCHM:	Single Chip Heterogeneous Multiprocessor
SE:	Standard Error
WPM:	Words Per Minute
NVS:	New Vocabulary Set
IVS:	Input Vocabulary Set
TDP:	Thermal Dissipation Power

1 Introduction

Conventional forms of parallelism are showing diminishing performance gains at algorithmic complexity levels that are well short of those found in the human brain, even as increasing architectural complexities are available. In contrast, the human brain seems to have the ability to maintain performance against almost unbounded growth in cognitive complexity, as learning continues and new memories are formed throughout our lives. For example, popular speech interfaces for smartphones struggle with providing accurate real time recognition beyond a few thousand words. This, even as they rely on mammoth cloud based solutions with hundreds of cores that consume thousands of watts of power [1]. Compare this to the human brain that handles vocabularies to the tune of 20,000 to 40,000 words (in just one language) [2], and offers consistent real time performance. Even as it multiplexes many cognitive tasks across limited cortical real estate (6 layers) and consumes only ~25 Watts of power [3]. We believe that the brain draws this ability by contextually partitioning its complexity recursively, essentially reorganizing itself to adapt to human behavior and simplifying its operation in order to facilitate survival. We observe that such reorganization entails extensive internal analysis that cannot proceed in the presence of external stimulus or else external information will be lost during this reorganization. Notably, the brain is marked by an absence of external stimulus while asleep, and is instead stimulated internally by a replay of existing memories which gives rise to dream imagery [4]. Based on the knowledge that dream sleep is essential for efficient cognitive performance and learning [5], we propose that it is also responsible for the contextual partitioning and reorganization in the human brain. Inspired by these observations, we design and evaluate a computer architecture for personal mobile devices, that implements contextual partitioning in a novel dream phase of operation.

The chief characteristic that our target applications must share is viability towards contextual simplification of their complexities. This holds true for applications for which multiple potential answers can exist, even for the same input, and the correct answer is dependent upon the user's context. Candidates include human computer interfaces and applications mimicking human intelligence where complexities can be personalized to the user. Some examples are speech recognition, gesture control, image recognition, personalized search tasks and pattern recognition, amongst others. A feature common to these applications is that their algorithms are trained on knowledge databases of symbols. For example, vocabulary units for speech, facial features for face recognition, patterned structures for pattern recognition. The size of the knowledge database reflects algorithmic complexities and is a determinant of performance in that we thus refer to their combined set as *Database Dependent Applications*. The contents of the knowledge database and its access patterns are strongly influenced by user context, which presents an opportunity for simplifying operational complexities and improvement in performance. We believe that mobile computing provides proximity to user lifestyle and makes available enough contextual categories to accomplish constant real time performance against rising algorithmic complexities. With every passing generation, as mobile computers become more pervasive, an even larger set of database dependent applications will emerge, which justifies their treatment as a separate class of applications in this research.

The design of database dependent applications is driven by the aim of evolving smartphones into cognitive phones [6]. A striking feature of the targeted cognitive tasks is that intelligence in them is strongly correlated to the size of the corresponding memory database, while the only performance requirement seems to be adherence to real time levels. For example, a strong measure of language proficiency is the size of the vocabulary, while recognition at rates any faster than spoken speech is unnecessary. Thus, even as the human brain dedicates a greater portion of its cortical area to support larger memory databases for vocabulary, patterns or faces, it does so, only to maintain constant real time performance. This observation frames the overall performance goal in this research effort as:

"*Constant real time performance against ever increasing algorithmic and architectural complexities, for database dependent applications*".

Figure 1 illustrates this performance goal and compares it to the performance of traditional methods of computer parallelism against a rising number of Processing Elements (PE) and algorithmic complexities denoted by A_n (where $A_{n+1} > A_n$).

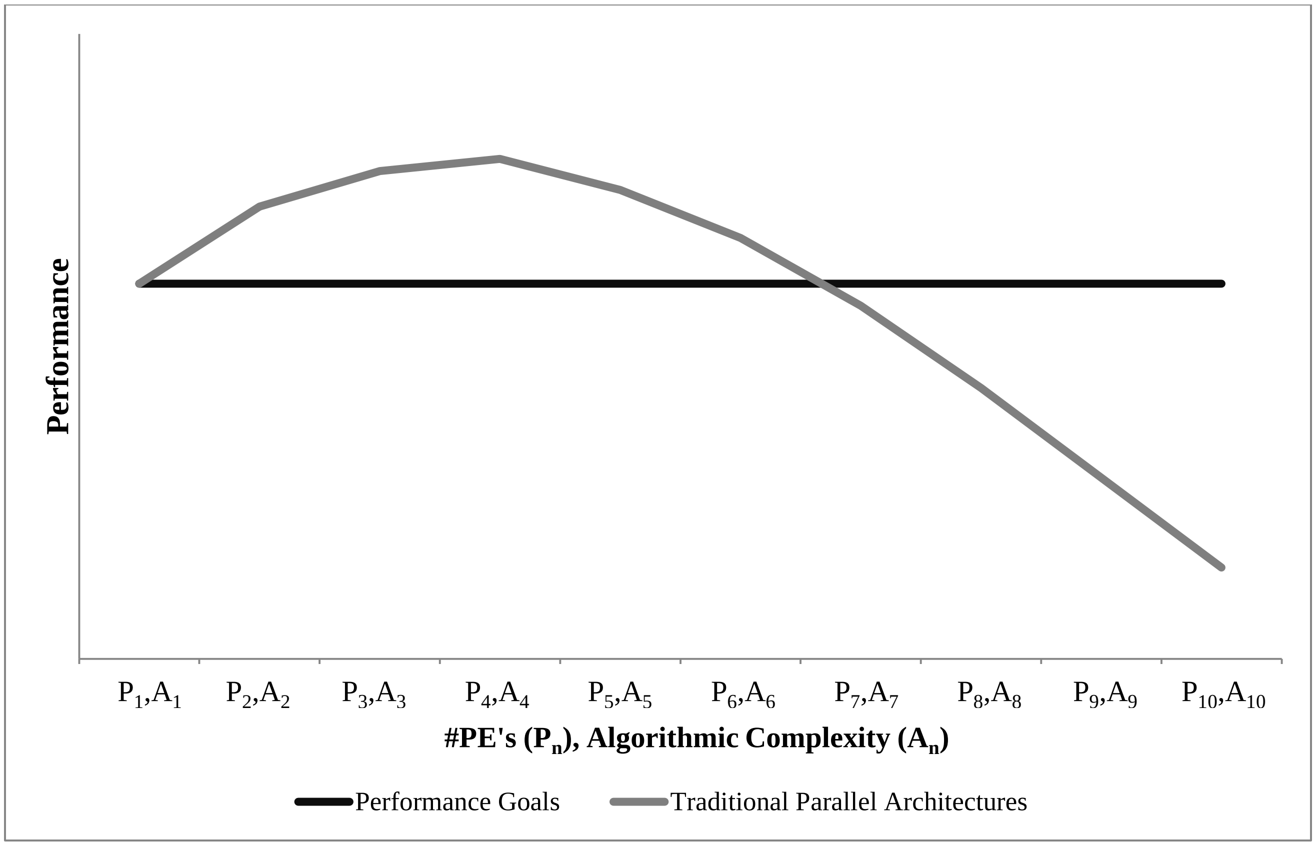


Figure 1: Comparing Performance Goals with Traditional Parallel Architecture Performance

Our goal, illustrated In the “Performance Goals” curve depicts how, as applications scale to brain like algorithmic complexities (i.e. A_{10} approaches vocabulary sizes in excess of 50,000 or tens of thousands of gestures or faces), and a proportional number of processing elements (1000’s) are used to support it, performance remains constant. This goal is radically different from the performance of traditional methods of parallelism, shown in the “Traditional Parallel Architectures” curve. For these traditional methods, as the system complexities (algorithmic and architectural) rise, initially performance improves alongside, and can even show a near-linear slope. The performance improvement slows down as the number of PE’s approach 4 or 5, and almost plateaus as the number of PE’s rise to 7 or 8. This is because these methods expose limited degrees of parallelism, thus there is an inherent need for synchronization, communication between parallel entities (threads, instructions). This results in large overheads that scale with system complexities. As a result, when architectures scale beyond 7-8 PE’s and proportional algorithmic complexities, performance begins to deteriorate [7-9]. Extrapolating this performance trend to thousands of PE’s and proportional algorithmic complexities, performance will degrade well below acceptable real time levels. Computer architects admit that except for a few embarrassingly parallel applications, exploiting thousands of threads and processing cores seems to be impossible with traditional methods [10]. Thus, our performance goals are unachievable with these traditional methods. Based on this observation, the overall aim in this research is to develop algorithmic and architectural solutions to achieve the performance goals shown in Figure 1. Specifically, in current research, we make two major contributions:

- Developing a Malleable approach for Contextual Partitioning (MCP)
- Conceptualization of the Dream Architecture for Lateral Intelligence (DALI)

A Malleable approach for Contextual Partitioning (MCP), inspired by the way human recognition and decision-making seems influenced by contexts, is developed as an algorithmic foundation for constant real time performance over growing algorithmic complexity. This draws on previous work that has shown the benefits of Contextual Partitioning (CP) in simplifying speech recognition training, resulting in improved accuracies and reduced training times. While our MCP approach can potentially simplify application operation, we also note that its benefits are coupled with overheads that will scale along a super-linear curve with algorithmic complexities. Existing parallel architectures however are limited in their scalabilities, and thus cannot allow MCP to achieve the desired constant performance. To address this, we conceptualize a novel architectural framework-the Dream Architecture for Lateral Intelligence (DALI) for performance evaluation of the MCP approach. The DALI draws on a recently proposed area of research-Computational Dreaming (CD). Computational Dreaming conceptualizes the implementation of dreaming in computers for the improvement of architectural performance. The experimental investigation is then setup to evaluate the DALI's performance for an example implementation to speech recognition. The results prove that the DALI in support of MCP can maintain constant real time recognition, as vocabulary sizes continue to rise and are partitioned over an increasing number of PE's, in this way achieving the performance goals shown in Figure 1.

The discussion proceeds along the following chapters: Chapter 2 discusses the background for our research and related research efforts in domains including Brain Architecture, Database Dependent Applications with a focus on Speech Recognition, Parallel Computation and the state of the art in mobile computing architectures. In Chapter 3 we discuss previous work in contextual partitioning (CP), and identify initial challenges in developing the MCP approach. Chapter 4 discusses the solutions to the identified challenges, which pave the way for the MCP approach. In Chapter 5 we discuss the previous work in Computational Dreaming, and identify the initial challenges in the DALI's conceptualization, and Chapter 6 discusses the solutions to the identified challenges. In Chapter 7 we discuss the setup of our experimental platform, which simulates the DALI for an example implementation to speech recognition. Chapter 8 discusses the experiments developed and executed over the experimental platform. Finally, in Chapter 9 we present and analyze the results and compare them to the expected performance goals.

2 Background and Related Work

This section discusses the background for our research and summarizes related research efforts. The discussion is divided into the following sections: First, we discuss research that supports and inspires our hypothesis on the brain architecture, and explains its scalable performance against rising complexities. Second, we characterize and define our target application set-database dependent applications, and discuss the algorithmic background of our chosen example application-Speech Recognition. Third, we discuss the performance constraints of existing forms of parallelism and some reasons behind the chasm between parallel computing and brain level complexities. This discussion also includes limitations of some recent research efforts in parallel speech recognition. Finally we review the state of the art in mobile computing in terms of architecture, application design and available speech recognition solutions.

2.1 Brain Architecture

The key insight that motivates our research is the brains ability to maintain constant real time performance for most recognition tasks, despite unbounded and dynamic cognitive complexity growth. In fact, one metric of human intelligence is its ability to process increasing complexity in cognitive tasks while maintaining real time performance. For example, the size of vocabulary comprehended in real time is often used as a measure of his/her command over a language. However, unlike the human brain, modern day computing systems are unable to achieve constant real time performance even at comparatively negligible complexity levels. We research the system, architecture level paradigms in the brain, to establish a thesis that explains its performance.

The motivations for research in brain emulation have been classified into three broad categories [11], First, developing simulations to understand the operation of the brain at various levels of granularity, Second, emulating the neural structure and operation of the brain to develop artificial neural networks for domain specific applications and Third, researching novel computer architectures by replicating the massive parallelism and plasticity of the brain. Our approach to emulating the brain belongs to the third domain, for which, a high level, abstracted understanding of the human brain is sufficient [11]. Towards this end, we were particularly influenced by the top-down, reductionist approach adopted by Jeff Hawkins in explaining the brains structure and operation [12]. Drawing motivation from this work, we propose the possible architectural and operational characteristics of the brain that explains the aforesaid performance. Our thesis is that the brain derives this performance by contextually categorizing its complexities, which is accomplished in the dream sleep. In the following sections we discuss the research efforts in support of this thesis.

2.1.1 Contextual Categorization in the Brain

Human memory for essential cognitive tasks such as speech, vision, and smell is categorized into Episodic Memory and Semantic Memory [13]. Episodic Memory (also called autobiographical memory) resides chiefly in the Hippocampus and consists of personal contextual experiences. Researchers [14] describe it as the memory of temporally dated or spatially marked episodes or events, and their temporal-spatial relationships. On the other hand, the semantic memory resides primarily in the neo cortex and consists of factual information, conceptual knowledge that can be consciously retrieved. Semantic knowledge develops by the gradual abstraction of common contextual features from personal memory streams (episodic memory) [15]. Episodic memory can thus be viewed as a contextual map of semantic knowledge. This map allows contextual cues to constrain semantic retrieval options, reducing operational complexity and leading to faster memory processing [15]. Some research [16, 17] points to the importance of such contextual categorization of input stimulus in the acquisition and use of skills such as language and cognition in general. It is thus apparent that contextual division and organization is central to the management and development of cognitive complexity.

Based on these observations we believe that the brains intelligence and performance efficiency stems from the ability to classify its growing memory database into ever more subtle categories. This allows the brain to continue to learn, while keeping its operation simplified by constraining its memory retrieval search space to a context specific category. As a consequence, a brain that is well learnt along a particular cognitive task (i.e. it carries a large memory database and maintains desired performance) will have a larger number, and more subtle contextual categories. This is consistent with the general model of intelligence, where a key indicator of learning is the knowledge of subtleties [18]. For example, between a music virtuoso and a regular music enthusiast, the former will be able to identify finer chords, pitches, whereas the latter might even struggle with identifying individual instruments. A similar classification of external stimulus into a hierarchy of groupings has been suggested to form the basis of the cortical operation in recent research [19]. In the next section we discuss the contributions of sleep and dreaming to such contextual classification and memory consolidation in general.

2.1.2 Dream Sleep and Memory Management

Irrespective of the pressures of daily schedule and against all survival instincts, every intelligent being must sleep or risk degeneration in the cognitive skills and processes [20-22] . Sleep is an unconscious state marked by an absence of external inputs to the brain, and consists of two alternating phases: Non Rapid Eye Movement (NREM) and Rapid Eye Movement (REM) sleep. The human brain dreams through both these phases [23], however NREM dreams have a larger proportion of recent memory and episodic content than REM dreams [24]. Dreaming is marked by a reactivation and replay of stored memories in the neo cortex and hippocampus [25]. The interruption of such replay has been known to impair memory retention [26, 27]. The nature of electromagnetic waves encountered during the two phases holds evidence for a strong dialogue between the neo-cortex and the hippocampus [28]. [29] suggests that such dialog supports the transfer of memory from the hippocampus to the neo-cortex and consolidates existing neo cortical memory.

The specific purpose and nature of dreams in REM and NREM sleep continues to be a highly debated area of research. However, scientists [30] agree that the neo cortical and hippocampal replay during the dream sleep is crucial for the consolidation and efficient recall of semantic memory. From these research efforts, it seems apparent that dream sleep is essential in the development and management of cognitive complexity. As discussed in the previous sub-section, contextual categorization underlies efficient cognitive operation and learning. Connecting these two observations, it is our thesis that the brain maintains its performance against rising complexities by contextually categorizing them in the dream sleep to enable simplified real time operation. In line with this thesis we explore Computational Dreaming as an avenue for achieving the desired constant performance (Figure 1) against increasing algorithmic complexities.

2.2 Database Dependent Applications

To guide our research effort we establish a classification in personal mobile computing applications to define our target set. We refer to this classification as Database Dependent

Applications, which includes natural human computer interfaces and user centric applications such as speech, image, pattern, face, gesture recognition, and personal mobile assistants. These applications are at the vanguard of research in ubiquitous computing, with the aim of making computers a seamless part of the human environment, to approach the ideal Invisible Computer [31]. Algorithms for these applications rely on a knowledge database, which consists of symbols that the algorithm has been trained on, such as vocabulary, facial features, and gestures. The performance of these algorithms, both in terms of accuracy and processing time, is directly dependent upon the size and complexity of the database contents. Thus, the knowledge database effectively represents their algorithmic complexities, which justifies the name used for this classification. These databases are also characterized by being incomplete, thus requiring inference in order to produce the best answer in the absence of a known correct answer.

Because these applications interface directly with the human users, they must provide intelligent, real time performance. However, for achieving true intelligence it is imperative that they support knowledge databases i.e. algorithmic complexities of the order of the human brain. As we will discuss in the upcoming section, the limited scalability of conventional forms of parallelism constrains the possible complexities these applications can carry. The key insight that drives our research is the strong relationship between the development and access patterns in the knowledge database with user context. The DALI uses this dependence to exploit a heretofore unexplored opportunity for the simplification of system complexity by partitioning the database using user context. For example, consider a real time face recognition application. The user can be expected to be around family/roommates when at home, his colleagues in the office, and the faces observed will vary accordingly. Partitioning them into these two categories will then allow the application to use only one of them based on the user location.

Amongst the host of other database dependent applications, we chose speech recognition for an example implementation in our present research. The reasons for choosing speech recognition were: First, it is a major thrust area of research and development in natural human-computer interface design, however decades of research has not made real time, accurate recognition at high vocabulary levels possible [32]. Second, it is a perfect candidate for the DALI as its use is naturally contextual, with vocabulary size being the chief determinant of its performance both in terms of accuracy and decoding time. Third, as a result of its status as a premier area of research, numerous open source tools and platforms exist that allow for development of speech recognizers. In the following section we discuss the background for the speech recognition algorithm, discussing its characteristics and implementation constraints.

2.2.1 Speech Recognition Background

Our discussion on the background of speech recognition is broken down along the two directions along which the research in this domain traditionally proceeds: First, we discuss the principles of Acoustic Model training and some research efforts that target its acceleration. Second, we discuss the characteristics and constraints of speech recognition algorithm.

Acoustic Model Training

There exist two principle classes of speech recognizers the Neural and Statistical [33]; whereas the statistical recognizers are based on Hidden Markov Models (HMM) the neural models utilize artificial neural networks (ANN). Statistical recognizers offer much better handling of input speech variability and have become the standard for most research and industry applications [34]. A Hidden Markov Model (HMM) represents units of speech using a network of hidden states. Each state is represented as a Gaussian mixture model and the weight of interconnections between the states represents the probabilities of transition [35]. The unit of speech chosen depends on the size of the vocabulary and the expected accuracy. Researchers suggest using larger units such as words to model small vocabulary sizes [35, 36]. However, with an increase in vocabulary sizes the accuracy and training times for models with larger units becomes unwieldy. On the other hand smaller units like phonemes provide higher accuracy with larger vocabularies, but it comes at the cost of increasing HMM size and longer training times. Triphones which can be considered to be contextually connected phonemes are the most commonly used units, as they lie between phoneme and word based models, and thus provide an optimal trade-off between accuracy and model size [35].

Statistical speech recognizers develop an acoustic model to represent the contents of the user’s dictionary. It is developed by training HMM’s using statistical information extracted from training speech samples. The statistical information is used to calculate the Gaussian mixture for each state and the inter-state probabilities. Acoustic model training is a computationally complex and time consuming process. The acceleration of this training phase to reduce time consumption while maintain accuracies has been the focus of many research efforts.

Currently popular speech recognition techniques rely on cloud computing to serve real time recognition requirements. However, because of reasons discussed subsequently in sub-section 2.4.3 we believe that local mobile device based recognition should be the preferred means of developing a truly natural speech human-computer interface. To this end, previous work in our group [37] examines the feasibility of partitioning user vocabulary contextually to reduce the training work load. This research shows benefits over traditional non partitioned methods along metrics of the number of training speech samples required, overall time spent training and easier adaptation to dynamic changes in vocabulary, while maintaining equivalently high accuracies. More importantly it achieves this without expending large amounts of architectural complexities [38-40], and is thus a perfect fit for mobile computer implementations. This research sets the precedent for our work, and allows us to assume the availability of accurate contextually divided recognition models for use in our research.

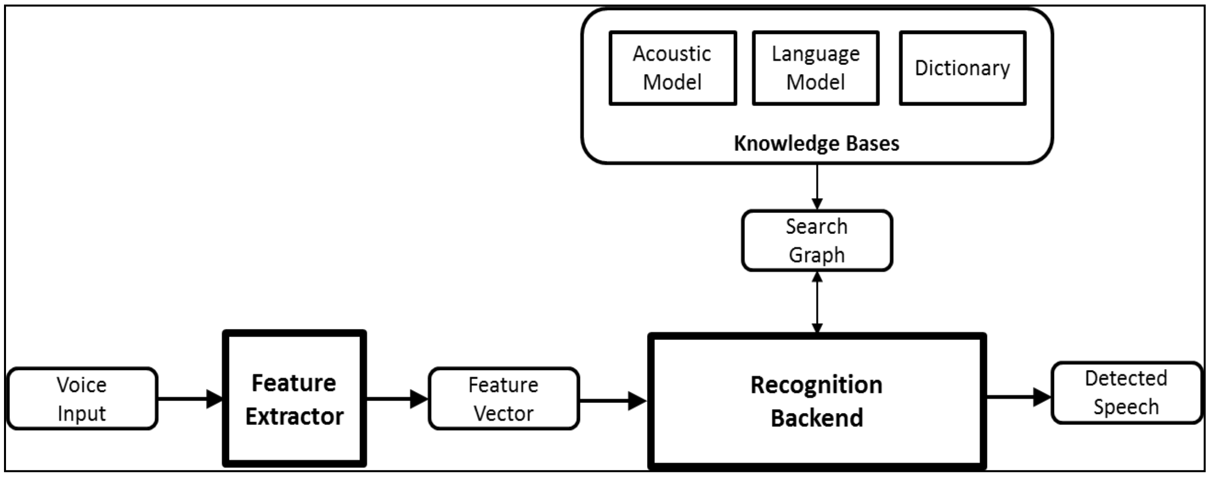


Figure 2: Speech Recognizer Components [41, 42]

Recognition Algorithm

As shown in Figure 2 [41, 42], a speech recognizer consists of a front end speech feature extractor and the recognition backend. The speech feature extractor extracts statistical information from the speech input to develop a feature vector that can be used for recognition. The recognition backend uses 3 knowledge bases [41]: First, the acoustic model developed through training. Second, a dictionary that provides pronunciations of words in terms of the constituent acoustic units (phonemes, triphones). Third, a Language (Grammar) Model that represents the probabilistic transitions between words based on the Language/Grammar. These knowledge bases combine to make a search graph, as shown in Figure 3 (adapted from [41]). Each state (node) in the search graph represents components of the Acoustical Model (HMM states), Dictionary (Triphones/Phonemes) or Language Model (Words) and the weight of the inter-state arcs represents probabilities of state transitions.

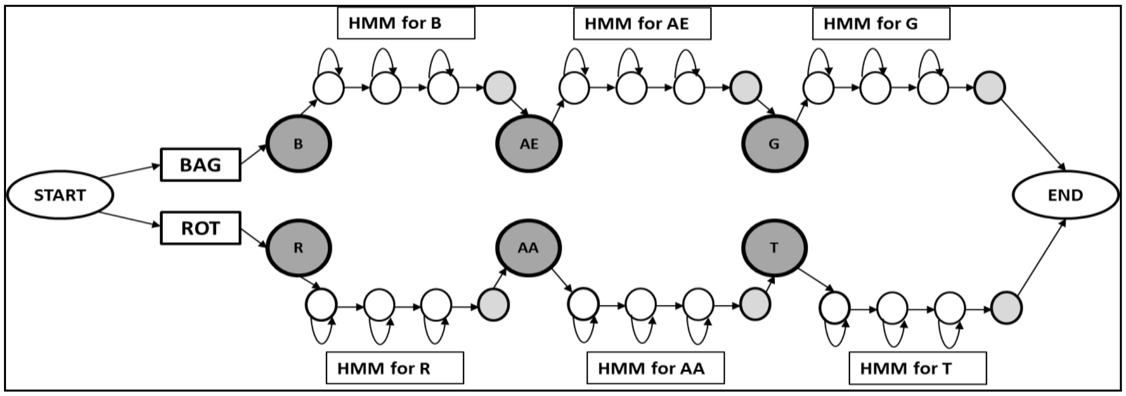


Figure 3: Speech Recognition Search Graph (Vocabulary: BAG (B AE G), ROT (R AA T)) [41]

Most speech recognizers implement graph traversal using Viterbi Iterations, where an iteration operates on a set of active states in the search graph and computes the set of active states for the next iteration. Each iteration consists of two phases [42]: the first phase collects the active states generated by the last iteration and calculates observation probabilities for each state. This is achieved by comparing the audio frame from the feature vector with the Gaussian mix of the active state. The second phase generates a set of active states for the next iteration by traversing through the search graph. It combines the interconnection weights with calculated observation probabilities to picks the states with the highest probabilities of occurrence.

Graph traversal in the recognition backend represents bulk of the computational complexity in speech recognition, and its acceleration using different forms of parallelism is an active area of research. However, researchers accept that parallelizing graph traversal algorithms a formidable challenge [7], because of data driven computations, unstructured graphs, poor localities (temporal and spatial) and high data access to computation ratios. The memory behavior of speech recognition algorithms in particular is extremely poor in comparison to other SPEC benchmark applications [43]. Growth in the vocabulary size further adds into the size and complexity of the search graph, exacerbating memory performance. Thus, the performance of speech and other similar algorithms is constrained by memory, limiting the efficacy of the available methods of parallelism, as will be discussed in the next section.

2.3 Parallel Computer Architecture

Traditionally, computer architectures are classified according to the Flynn's Taxonomy [44], based upon their parallel handling of input, output data streams. Architectures are divided into 4 groups: Single Instruction Single Data (SISD), Multiple Instruction Multiple Data (MIMD), Single Instruction Multiple Data (SIMD) and Multiple Instruction Single Data (MISD). SISD architectures correspond to traditional single core architectures, where Instruction Level Parallelism (ILP) has been the dominant method of performance improvement. Through the 1980s and 1990s, ILP improved performance of single core architectures by approximately 52% per year. Around 2003, single core architectures hit a power and complexity wall and the performance benefits from ILP have since diminished, slowing down uniprocessor performance improvement by almost 5 times [8]. Since then, the focus has been on MIMD architectures that leverage Thread Level Parallelism (TLP), fueling the growth of multicore architectures. SIMD architectures have always had a niche market limited to computation intensive tasks, especially those that are oriented around structured data such as that found in linear algebraic applications. A common implementation is leveraging data level parallelism (DLP) in graphic processing units (GPU). Recent research has begun to leverage GPU designs for some data intensive personal computing applications. By contrast, MISD architectures are not well defined, and have generally been associated with fault tolerant computing which requires redundant computation. Another interpretation for such an architecture, where different computational units work on the same data is in pattern recognition tasks where multiple models work on the same pattern to produce an answer [45]. Still others consider MISD computing to be almost nonsensical, as a single data stream would result in a multitude of different, but potentially simultaneously correct, answers.

Presently, in personal computers, TLP continues to be the dominant form of parallelism, even as researchers are unable to scale beyond 6-8 PE's for most common applications [8, 9]. DLP with GPU's has made some inroads into personal computing applications, as researchers attempt to leverage heterogeneous computing. Both TLP and DLP however have exhibited classic limitations in automatically finding parallelism at the granularity of a thread or data level computation, and have found scalabilities limited. Here we discuss these limitations in detail by reviewing related research efforts in TLP and DLP. Subsequently, we focus on the research efforts in parallel acceleration of our target application-Speech Recognition, and their limitations.

2.3.1 Data Level Parallelism (DLP)

DLP is applicable to algorithms that are characterized by repetitive data manipulation instructions and has traditionally been limited to applications from the scientific or niche domains like graphics processing. However, this has changed with the development of programming models such as OpenCL and CUDA that make it easy to program Graphics Processing Units (GPU) for general computing workloads [46, 47]. Referred to as General Purpose Graphics Processing Units (GPGPU), researchers are beginning to exploit the data intensive components of general algorithms by distributing operation over a GPU and CPU. The GPU runs the compute intensive component of the algorithm, and the comparatively scalar components are handled by the CPU [46]. GPGPU designs have found intensive use in the High Performance Computing spectrum [48], and have shown excellent performance per watt results [49].

However, apart from and some compute intensive general purpose applications most personal computing applications show little DLP and are instead limited by memory accesses. GPGPU architectures have thus failed to make significant inroads into personal computing applications. Further, GPU performance is constrained by the unavoidable workload transfer between the CPU and the GPU, because of the limited bandwidth [50]. Researchers agree that this constitutes a major performance bottleneck [47], and show that an inclusion of the workload transfer overheads makes a significant negative contribution to the originally reported GPU speedups [51]. While the introduction of fused GPU-CPU architectures have shown some improvements over discrete architectures by avoiding PCIe transfers, the overhead still scales almost exponentially with increasing workload sizes [52]. Thus, apart from a niche application set that is marked by high levels of DLP, GPGPU’s seem to be unable to provide scalable performance benefits against increasing algorithmic complexities.

2.3.2 Thread Level Parallelism (TLP)

In the early 2000's TLP helped architects avoid the impending power wall and showed early performance gains. This lead to some architects suggesting that the software designers should start developing for architectures with hundreds to thousands of cores [53]. But such optimism was short lived, as extracting performance benefits from TLP has proven to be non-trivial. First, physically the amount of task level parallelism present in an application is limited, and as per the Amdahl's Law even small fractions of serial component in applications diminishes the performance gains of Parallelism [54]. Secondly, Unlike ILP which extracts parallelism dynamically using a combination of hardware structures and intelligent compilers, TLP delegates the responsibility of identifying task level parallelism to the software designers. Furthermore the designers need to make careful choices of the compiler, scheduler and the hardware platform in order to fully exploit TLP. This results in considerable software redesign and effort spent on extracting minute amounts of parallelism [55]. Thirdly and most importantly, the performance of TLP is limited by the unavoidable overheads of communication and synchronization. These overheads scale exponentially with rising thread/core count, further overshadowing the performance benefits of TLP [56]. The cumulative effect of these issues is that while the industry leaders produce architectures with a rising number of cores [57], the performance benefits of TLP architectures fail to scale beyond a miserly 3-4 cores for most personal computing applications [8, 9].

Both TLP and DLP are hindered by inherent overheads. The algorithmic and architectural complexities of these methods are tied closely together, causing overheads to scale nearly exponentially with complexity growth. Thus, traditional methods of parallelism might show some performance benefits at low complexity levels; however they are quickly overshadowed by the overhead costs as complexity increases. The performance of these methods is especially bad for search graph based algorithms, which includes most database dependent applications [7]. Altogether, this discussion justifies the degrading performance curve plotted for traditional methods of parallelism in Figure 1, and means that conventional methods of parallelism will continue to limit database dependent applications to complexity levels well below that of the human brain. Next, we review research in parallel acceleration of our example application-Speech Recognition.

2.3.3 Parallel Acceleration of Speech Recognition

The three chief methods that have been leveraged for parallel acceleration of speech recognition have been Thread Level Parallelism (TLP), Hardware Acceleration and Data Level Parallelism (DLP) on both GPU's and General Purpose Graphics Processing Units (GPGPU). Here, we summarize related research efforts and their limitations in these three methods.

Thread Level Parallelism

Static partitioning of the search graph constituents amongst threads over a 4 cored PLUS microprocessor was explored with results showing a speedup of about 3.8 [58]. Despite efforts made to divide computation as uniformly as possible, the dynamic nature of path traversal leads to large load imbalances between threads, thus limiting scalability to 4-5 cores. Another research effort [59] uses a multiprocessor simulator to examine dynamic load balancing of the active search graph paths over a multi-processor system. Their results show an upper bound of 4.71 for parallel speedup, but they do not consider the overheads of synchronization, thread creation or constrained memory bandwidth, which will further limit the results. From these research efforts it is evident that the scalability of multiprocessor systems for graph traversal is limited by memory constraints that are inherent to database dependent algorithms. Their performance can thus only be expected to worsen with increasing vocabulary sizes and accuracy.

Hardware Acceleration

There were numerous early attempts at developing hardware acceleration techniques for speech recognition, but were unable to achieve desired performance. However, the need for satisfying the power constraints of mobile computing devices has led to resurgence in this research [60]. [61] explores a cheap and power efficient speech interface through a hardware implementation of an HMM based recognizer. While it does show power benefits over a low frequency single core ARM processor, no comparisons are made with a parallel multi core implementation. Further, the system requires exponential increases in memory size for small increases in vocabulary, thus limiting its ability to scale. Another research effort [62] couples an Intel XScale processor with a speech co-processor to optimize power performance of recognition on portable devices. The co-processor consists of multiple processing elements that use simultaneous multi-threading (SMT) to hide memory latencies. However, improvements in decoding time start to show diminishing returns after 4-5 cores. The authors recommend that future work focus on memory system optimizations to improve memory performance of the algorithm, as it continues to be a major contributor to both power and time consumption. It is clear that methods of hardware acceleration show benefits along power consumption, but continue to face the memory constraints inherent to speech recognition. Furthermore, hardware designs pose limitations on dynamic changes in the application algorithm, which are a staple of speech recognition algorithms as a result of changes in user vocabulary.

General Purpose Graphics Processing Units (GPGPU's)

Recent work in accelerating speech recognition has focused on using general purpose graphics processing units (GPGPU) for accelerating the Viterbi iterations in the recognition backend. This is a consequence of the emergence of supporting programming models such as CUDA and OpenMP that eased the programming of GPU's for general purpose computing [63]. Dixon et al.[64] port the computationally complex phase of observation probability computation (Phase 1) to the GPU, while executing the other phases over the CPU. However, the performance penalties imposed by the frequent and large workload transfers between the CPU and GPU limit the performance benefits of this approach. You et al. [65] minimize CPU-GPU communication by implementing the entire recognition process over GPU's. The performance benefits however, are limited by the memory size of the GPU, as with increases in model size CPU to GPU transfers become mandatory again. In an attempt to overcome the GPU memory constraint, [66] uses a simple language model on the GPU, and rescores the produced hypothesis using a complex language model in parallel over the CPU. They achieve higher speedups of around 11x over a sequential implementation, using a GTX680 NVIDIA GPU. Table 1 compares GPU architectures between those mobile device SoC's (GeForce ULP (Tegra 4 SoC) and GPGPU scale computers (NVIDIA GTX 680).

	NVIDIA GTX 680(GPGPU)	**GeForce ULP (SoC)**
SIMD Cores	1536	72
Performance (GFlops)	3090	74.8

Table 1: Comparing GPU architectures between GPGPU research and Mobile SoC's

From table 1 we observe that the GPU architectures on mobile SoC's are an order of magnitude weaker than those used in GPGPU research. Thus, the use of GPUs for general purpose computing seems to be viable for cloud or server based implementations of speech recognition, but is not a feasible alternative for mobile devices.

It is apparent that the performance of traditional methods continues to be limited by overheads of communication and synchronization for TLP or workload transfers for DLP (GPGPU's), with the results obtained approximating the diminishing performance curve as shown in Figure 1. The memory bound behavior of the speech recognition algorithm further worsens the scenario, which makes it impossible to scale to high vocabularies using personal mobile computer architectures while providing accurate, real time recognition. It is thus apparent, that our performance goals as shown in Figure 1, of achieving constant performance against complexities of the order of the human brain cannot be achieved with the traditional methods of parallelism, justifying our investigation into novel forms of parallelism.

2.4 Mobile Computing

The demand of mobile computers (smartphones and tablets) has shown a steep rise through the past decade, while the traditional personal computer demand seems to have hit a plateau [67]. Their growth has been fueled by their small, portable form factors and diverse software ecosystem that provides applications for virtually every facet of life. The development of novel, unobtrusive human-mobile interfaces is an active area of research [68, 69]. This will further accelerate the seamless integration of smartphones into everyday life, consequently becoming the interface of users to the external world and developing into virtual avatars of the users personas. Scientists suggest that as phones begin to learn lifestyle patterns and user behavior to offer user centric performance, they will evolve into cognitive phones [6]. In the following sections we present a discussion of the latest trends and state of the art in mobile computing design. Our discussion is broken down along the two directions in which the research and development in mobile computing has proceeds i.e. architecture and applications design. Following these sections we discuss the state of the art applications available for personal computer speech recognition, and assess their limitations.

2.4.1 Present Architectures and Future Trends

Mobile computer development is driven by requirements for small form factors, high performance across a myriad set of applications, long battery lives and fast time to market.

System-on-Chips which are highly integrated packages of multiple heterogeneous processing elements have powered the mobile computers through the past decade. Their popularity is a consequence of: First, high IP reusability that allows fast time to market, Second, low interconnection overhead and small form factors because of single package integration, Third, heterogeneous processing elements that provide for the diverse processing requirements of mobile applications. The heterogeneous set of processing elements usually includes a multicore central processing unit (CPU), graphics processing unit (GPU), Audio/Video signal processors and a Modem. Table 2 summarizes some currently popular SoC's for smartphones and tablets. As process technology improves, designers have used the extra real estate to add more processing elements on to the chip. According to ITRS [57], this growth is expected to continue with the chips expected to carry upwards of 1000 cores in another decade.

	Apple A6	**Exynos 5 Octa**	**Tegra 4**	**Snapdragon 600**
Clock	Up to 1.3 GHz	1.8 GHz-A15 + 1.2 GHz-A7	Up to 1.9 GHz	Up to 1.9 GHz
CPU	Dual Core Swift	Quad Core A15 + Quad Core A7	Quad Core A15 + Power optimized A15	Quad Core Krait 300
Devices	Apple IPhone 5	Samsung Note 2, Samsung S4 (Asia)	Vizio Tablet, HP Slatebook x2	Samsung S4 (USA), HTC One
Process (nm)	32	28	28	28
GPU	PowerVR SGX 543MP3	PowerVR SGX 544MP3	NVIDIA Ge-Force ULP	Adreno 320

Table 2: Popular Tablet and Smartphone SoC architectures [70, 71]

Early mobile SoC designs featured multiple homogeneous CPU cores; however recent designs have adopted heterogeneity in the design of CPU's. For example, Exynos from Samsung pairs powerful but power hungry ARM A15's with power efficient and simple ARM A7's, and NVIDIA's Tegra 4 pairs A15's with a low frequency power optimized companion core. The use of single ISA heterogeneous cores allows designers to exploit the variability in processing requirements across and within applications. This is achieved by migrating workloads between the cores, which helps mitigate Amdahl's Law and provides twin benefits of improved throughput [72] and power performance [73]. Heterogeneous system on chip architectures continues to be an active area of research, and the future architectures can be expected to carry many heterogeneous core types.

The on chip interconnect mechanism in SoC’s is an essential design concern, as it effects the power and area consumption in the chip extensively [56]. Traditionally SoC’s used shared medium interconnects, where the processing cores need to arbitrate for the shared bus to communicate with other nodes. Some examples for such interconnects are earlier iterations of AMBA from ARM, CoreConnect from IBM. While they serve the needs for current systems that have 4-5 communicating nodes, researchers [74] agree that they are not a scalable solutions for future designs with 100’s of processing cores. Network-on-Chips (NoC) [75, 76] have been proposed as scalable alternatives to the traditional designs. NoC borrows principles from the design of Wide Area Networks, in decoupling the physical and logical transport of messages, and reliance on packet switching for message delivery [77]. This allows NoC’s to support an arbitrarily large number of PE’s while ensuring quality of service, better power performance, higher bandwidths and enhanced modularity in design [78] [77]. Further, unlike wide area networks the system characteristics and components are known at design time, which allows for tailored designs that cater to user and application needs [79]. Network on chips have started to replace existing interconnects in commercial SoC’s , and are expected to be the dominating means of interconnection in the near future [80].

2.4.2 Mobile Applications and Context Aware Design

Mobile phones have evolved from their original role as pure communication devices to portable general purpose computers. This evolution has been a consequence of the availability of intelligent, interactive applications that use the wealth of programmable embedded sensors on the mobile devices [81]. Intelligent, user-centric applications now exist across a myriad set of domains such as social networking [82], healthcare [83], and intelligent transportation [84]. Mobile designers focus on developing seamless interaction mechanisms with the end users, to develop the ideal invisible computer [31]. Towards this end context aware mobile computing has been a major area of research, it aims to leverage the close relationship of user behavior with context to provide context optimized computing. Research efforts have developed methods to compute the user context from sensory information. Some examples are the Mobile Sensing

Platform [85], a wearable context recognition device that uses multiple embedded sensors like the accelerometer, barometer, photo sensor, and microphone. Similarly, [86] provides a user activity recognition framework based on GPS location trace analysis, and [87] develop a context recognition, activation system using cues from multiple low level sensors for mobile phones and PDA's.

It is thus possible to develop and utilize user context accurately in mobile computing, and further improvements in the quality, quantity and diversity of embedded mobile phone sensors can only improve the accuracy and specificity of context prediction. However, heretofore the use of context has been limited at the software level, as a tool for presenting context specific information or tagging it for later retrieval [88]. In contrast, our research explores context as a hardware-software solution for providing an optimized reorganization of the algorithm complexity over architectural complexity to simplify system complexity.

2.4.3 Speech Recognition on Mobile Phones

While mobile computing devices have laid the foundation of ubiquitous computing, designers acknowledge the need to improve human computer interaction for seamless integration into user lifestyle. Speech, because of its natural usage as a form of communication provides an excellent interface for improved human computer interaction. As discussed in the previous section large vocabulary sizes impose high computational complexity in speech recognizers. There exist numerous mobile computing solutions for speech recognition, most popular amongst which are Siri from Apple, Google Now (Voice to Text) from Google and Dragon Dictate from dragon technologies. However, owing to the complexity of the speech recognition with large vocabularies, all these solutions use cloud services to run the recognition process. Cloud services offload the application to a remote server (cloud), and exchange data with the server to process inputs. Numerous software companies have either successfully implemented cloud versions of their popular applications, or are expected to do so in the future.

While cloud based speech recognition is an active area of research, we see the following two problems that limit its application as a natural human-computer interface: First, relying on mobile internet for processing requires consistent internet connection and a stable bandwidth, which is a tough proposition to bank upon [89]. Future mobile devices can be foreseen to carry multiple database dependent applications such as pattern, image, and speech recognition. These applications will execute concurrently and compete for the internet bandwidth with tens of other personal computing applications such as word/email processors, music applications, and games. In this scenario any inconsistencies in the internet connection would defy the very purpose of an intelligent human computer interface application like speech. While internet services will continue to improve in the future, offering higher bandwidths with greater fidelity, the user demands from both applications and devices will also scale equally, if not faster. Second, sending personal speech data over the internet poses a serious risk to the security of a personal mobile computer and consequently the user [89]. Personal data security in the increasingly connected world is an issue of great concern, and relaying sensitive information like speech samples that might contain personal information would stand in direct violation of these concerns. Considering both these issues, it is our belief that intelligence of mobile devices must stay local to the device, while applications such as word processors, music players and image editors should be ported out to the Cloud. It is thus essential that personal mobile computer architectures be developed to process speech and other intelligent database dependent applications locally.

2.5 Summary

In this chapter, we discussed the background and related research efforts for our investigation. The key conclusions are that increasing pervasiveness of mobile computers is fueling the growth of database dependent applications that attempt to imitate human intelligence. However, brain level complexities seem unapproachable using existing methods of parallelism, as their scalabilities are limited to negligible algorithmic-architectural complexities compared to the human brain. Currently, database dependent applications execute over remote cloud servers, however this trend cannot be sustainable, because of power, setup costs and security issues. Thus, an investigation into novel forms of parallelism is essential for achieving brain level complexities on mobile devices.

3 Contextual Partitioning (CP): Previous Work and Challenges

Research in Contextual Partitioning (CP) is inspired by a hypothesis that the human brain uses context to simplify cognitive complexity, as complexity continues to grow over time. Previous work [37] has established its benefits in optimizing the training of speech recognition algorithms for static vocabulary sizes. To achieve our goal of constant real time performance against rising algorithmic complexities (Figure 1), the first step is to develop a CP approach that provides scalable performance benefits. CP draws benefits by personalizing applications to user behavior, with partitions showing performance benefits in proportion to the level of personalization [37]. Thus, we must be able to personalize partitions ever closer to human behavior. An inherent characteristic of human behavior is its flexibility, manifest in varying routines, habits and one off digressions that result in inconsistencies along contextual patterns, even those that are well established. These inconsistencies must be encompassed for closer personalization, which is possible only with an approach that accommodates malleable partition definitions, inspired by human behavior. Consequently, as the first major contribution of this research we develop a malleable approach to contextual partitioning. We identify three initial challenges in this investigation:

- Origination of partitions.
- Integration of partitions.
- Retirement of partitions.

An apparent impact of behavioral flexibility is on the lifecycle of contextual patterns. As newer patterns *emerge*, while some existing patterns are *sustained*, and others become *redundant or disappear,* over the users lifetime. These define three fundamental phases in the lifecycle of a contextual pattern, and to ensure malleability, the partitions must be able to adapt with these phases. Towards this end, the challenges itemized above i.e. origination, integration and retirement of partitions, each targets an adaptation to one of these three phases respectively. Solving them will thus pave the way for a malleable approach in CP, and bring us closer to the desired levels of performance scalability. Here, we first summarize previous work in CP, and subsequently discuss these challenges.

3.1 Previous Work

Contextual partitioning (CP) draws inspiration from the handling of external stimulus by the human brain, where context seems to play an important role in decision making and reducing the search space for memory retrieval [17]. For computer systems, CP provides benefits by executing the same redundant algorithms, however over contextually partitioned data, consequently simplifying applications in real time. Previous work [37] has used CP to simplify the training for smartphone based speech recognition systems. The vocabulary of an example user personality worth 320 words is divided into 6 partitions, at two levels of specificity: Coarse and Fine grained. As the name suggests, fine grained partitions are personalized closer to user behavior than the coarse partitions. In comparison to an un-partitioned vocabulary database, the CP approach shows optimized training performance. Specifically, training with partitioned vocabulary databases takes reduced times while not compromising on recognition accuracies, and shows a higher robustness towards vocabulary changes. Significantly, fine grained partitions (i.e. the ones personalized closer to the user behavior), offer better performance, because of a greater simplification of user vocabulary.

A novel form of computer parallelism – Context Level Parallelism (CLP) is also introduced. In CLP, the PE's execute the same application algorithm; however each is allocated a distinct partition, in this way simplifying the algorithmic complexities. CLP presents an alternative, more scalable form of parallelism than traditional methods like TLP, for applications where a contextually pertinent output can be distinguished from multiple alternatives. This is because each partition can potentially process in absolute independence, and CLP can thus offer almost ideal scalability against a rising number of partition-PE pairs. This can allow for speech recognition local to the smartphone, even with large vocabulary databases. Traditionally, because of the resource constrained architectures of mobile devices, training of speech recognition has been implemented on the cloud. However, as discussed earlier, a localization of intelligent applications offers better security and avoids performance losses due to the internet bandwidth/fidelity issues. We now discuss the challenges faced in extending CP to achieve the performance goals shown in Figure 1.

3.2 Challenges

While the potential of CP in simplifying the operation of speech recognition has been established, its implementation is limited to small, static vocabulary sizes. However, to achieve our goal of maintaining constant real time performance against increasing algorithmic complexities, CP must continue to provide benefits even as algorithmic complexities approach brain like levels. For speech recognition, this entails tackling vocabularies upwards of ~50,000 words, whereas previous work was limited to about 320 words [37]. Performance benefits in CP result from classifying algorithmic complexities to match contextual patterns in user-device interaction, and activating them selectively in real time. Also, the closer a partition is personalized to user behavior, the higher are the achievable performance gains. Thus, a key requirement for scalable performance benefits with CP is the ability to adapt partitions closer to user behavior.

Human behavior is characterized by its flexibility, as habits, routines vary with changing circumstances, and in turn disturb even well-established contextual patterns. For example, the sleep cycle can vary radically when on vacation or when visited by parents, both temporary digressions. On the other hand, upon taking a job, a more permanent shift in sleep hours can be expected. The human brain is plastic in equal measure to this flexibility, even exhibiting the ability of re-programming cortical areas to prioritize between cognitive tasks or adjust to a loss of a sensory ability [90]. In the same vein, to adapt closer to human behavior and achieve scalability, it is essential that we develop a malleable approach in CP that can mirror this flexibility.

At a high level, one impact of behavioral flexibility is evident in the finite lifecycles of contextual patterns. In general, while some patterns will have short lives-even developed in error, others can last longer-even for a lifetime. For example, as a user leaves for vacation, patterns such as sleep hours, places frequented or mobile applications accessed will alter, however only for the duration of the holiday. On the other hand, as the user shifts employers, entirely new patterns might arise to replace the ones from the older job, possibly for the foreseeable future. Three fundamental lifecycle phases are apparent: new patterns can *emerge*, as habits are formed or routines established, existing patterns are *sustained*, and some older patterns might grow to be *redundant or completely disappear* from user-device interaction. To ensure malleability, the partitions must be able to adapt onto these fundamental phases while continuing to offer real time performance. For this, we are faced with the challenges of originating, integrating and retiring partitions to address the respective fundamental phases. We now discuss each of these challenges in further detail here.

3.2.1 Origination

Partitions originate when meaningful classifications in algorithmic complexities can be obtained using prevalent contextual patterns. Contextual patterns can exist along numerous sources of information and at many levels of abstraction. However, depending upon the user behavior and application characteristics, only a subset might provide meaningful partitions. Origination can thus entail extensive analysis of user-device interaction. For example, at the present moment, the readers context can be specified in terms of location between university or home, the device being used – a laptop or smartphone, time at granularities from the minute, hour, phase of the day or even higher, the application/website being accessed on the device and even the current cursor/mouse location. Inputs for an application-user pairing (e.g. speech recognition for a university student), might be biased strongly by location, and show minimal dependence on time of day, whereas for another application/user, it might be the opposite. This makes the originating partitions, without accruing unscalable amounts of time, a formidable challenge. Initially, we identify the following two problems in originating partitions:

- Scalability against rising algorithmic complexities
- Application agnosticism

Previous work in CP develops and utilizes a small set of partitions, as it optimizes training for fixed vocabulary sizes. Even previous work in context aware computing leverages only a handful of classifications, because research goals are limited to tagging, presenting information and automating applications [88]. However, if we are to approach algorithmic complexities at the level of the brain, a limited set of partitions will not suffice. This is because, as algorithmic complexities increase, the sizes of existing partitions will also rise and thus result in diminishing performance benefits. As an example, consider the speech recognition interface tackled by CP in previous work [37]. It partitions a vocabulary size of ~300 words into 6 partitions. A natural English speaker on the other hand is approximated to have a vocabulary of at least 40,000 words, with numbers being much higher for a multi-linguist user. Now as vocabularies grow to this level, the sizes of the 6 partitions will also approach thousands of words, and entail a proportional degradation in performance. Consequently, if performance benefits are to be maintained, the number of partitions will have to scale with algorithmic complexities.

Previous research efforts in CP and even context aware computing target individual applications, and develop application specific approaches to leverage context [37]. These approaches might be valid for originating partitions in the target application; however their cross application implementation is limited, because user behavior varies with application characteristics. Scientists acknowledge the need for a unified, application agnostic approach for leveraging context [88], however a popular methodology is yet to emerge. At the same time, as we target the set of database dependent applications, whose numbers are set to increase going forward, developing application specific approaches will quickly become cumbersome. In this light, in addition to scalable origination, we also investigate an application agnostic approach in originating partitions.

3.2.2 Integration

Following their origination, a partition will remain active until the corresponding contextual pattern is sustained in the user lifestyle. In this time period, as the user interacts further within the established patterns, new algorithmic complexities will be encountered. This poses the problem of integrating the new algorithmic complexities into appropriate partitions, by analyzing the accompanying contextual inputs. Integration into incorrect partitions will be detrimental towards real time performance, as chosen partitions would fail at producing the correct answer, and inputs would have to jump between multiple partitions or worse, go completely unanswered. Often however, it might be difficult to draw a strict correspondence between complexities and partitions, as some inputs might be received when the user is between distinct contextual situations, such as moving between floors or switching between applications. And, at other times complexities are not exclusive to a single partition, like a basic set of speech commands used across applications. As the number of partitions and the contextual situations they cater to increase, integrating new algorithmic complexities into accurate partitions can grow to be computationally complex. An approach to integration must address the following initial problems:

- Integration inaccuracies
- Minimizing overheads of integration

Integration into erroneous partitions will result in the chosen partitions being unable to produce accurate and timely answers. We refer to these errors as Integration Inaccuracies. Ideally, the aim must be an absolute elimination of these inaccuracies, however even if we assume an algorithmically accurate analysis, some inaccuracies will be unavoidable. These correspond to the inaccuracies that are inherent in user behavior, perpetuated by learning curves with new devices and applications. For example, while adapting to a new mobile device, users can often utilize incorrect spoken commands or at other times confuse commands between applications / menus. In these and other similar scenarios, user context will point to incorrect partitions, irrespective of the thoroughness of the algorithmic analysis. As scientists believe that such inaccuracies are an integral part of human learning [91], the integration inaccuracies can be expected to be a natural component of user-device interaction. While the performance losses of integration inaccuracies might not be completely avoidable, considering their inherence, we investigate to minimize their performance penalties.

Every new partition, especially ones that lie close in context will add to the search space for choosing candidate partitions for integration. Consequently, as partitions rise to thousands, the overheads for integration can rise exponentially alongside, severely affecting the scalability of contextual partitioning. While ensuring accurate integration is an important concern, it cannot come at the cost of our chief goal of scalability. We thus investigate to minimize the overheads of integration, while limiting the compromises on its accuracies.

3.2.3 Retirement

Much like their appearance, the eventual redundancy or even disappearance of contextual patterns is an inherent consequence of lifestyle variations and accompanying changes in user behavior. This would make the corresponding partitions also redundant, providing the opportunity for utilizing the algorithmic-architectural space that they occupy for more pertinent partitions. At other times, a pattern might not even completely disappear; instead its frequency of occurrence may not merit spending architectural and algorithmic resources to maintain the corresponding partition. Retirement of partitions however is challenging because the disappearance of a contextual pattern can only be calculated probabilistically. For example, even as the user returns from a vacation in San Diego, it cannot be guaranteed that he/she will not revisit it in the near future. At best, its disappearance can only be estimated by analyzing the frequency of past visits, time since last visited and other parameters that are once again, specific to the user. Retirement of partitions thus remains an interesting challenge to address, posing the following initial problems in this investigation:

- Retirement inaccuracies
- Maximizing the simplification in architectural-algorithmic complexities

Because the disappearance of a contextual pattern can only be gauged probabilistically, erroneous retirements are unavoidable. This can result into inaccurate real time operation, similar to the way intelligence does not result in a goal of perfection. For example: if the user travels back for another vacation to San Diego, and the corresponding partitions have been retired, inputs will go unrecognized. Over time, humans forget the nuances of places they have been, if those memories are not reinforced for daily need. We refer to these inaccuracies as Retirement Inaccuracies. As thousands of partitions are obtained, tracking each contextual pattern and making correct choices will become challenging, and potentially multiply the instance of retirement inaccuracies. While they cannot be completely avoided, if performance benefits are to be maintained, minimizing their performance penalties is essential.

The partitions are forced into retirement as a result of both algorithmic and architectural constraints. Architectural constraints exist because an unbounded amount of chip real estate is not available for allocation to partitions. Algorithmic constraints result due to overheads of integration that scale with the number of partitions, and can potentially limit the scalability of CP. As a growing number of partitions exert greater algorithmic-architectural pressures, even partitions that have only a low probability of having disappeared will be forced into retirement. This will increase the instance of retirement inaccuracies and the resultant performance penalties. Keeping the retirement inaccuracies in mind however, the retirement of partitions must be minimized as far as possible. Thus, it is critical that the simplification in the algorithmic-architectural complexities accompanying each retirement be maximized. This will essentially delay the next retirement, and consequently the probability of a retirement inaccuracy.

In this chapter we defined our first research challenge as the development of a malleable approach to contextual partitioning. To ensure malleability, we identified three initial challenges of originating, retiring and integrating contextual partitions. In the next chapter we discuss the solutions to these challenges.

4 Towards Malleable Contextual Partitioning (MCP)

In this chapter we discuss the first contribution made in this research: an approach for Malleable Contextual Partitioning (MCP). In the previous chapter, we have identified and analyzed three initial challenges in its development. Namely, these are the origination, integration and retirement of partitions. We now discuss the solutions developed for these challenges.

4.1 Origination

As the first step, we investigate the origination of partitions with scalability and agnosticism of the underlying application. A key insight that guides this investigation is that in approaching high algorithmic complexities, user context can hold widely different meanings. This is because algorithmic complexities are strongly coupled with new usage scenarios as are exposed by rising user-device interaction, and within each scenario, the contextual information that best classifies the user's behavior is bound to vary. Further, each database dependent application will potentially introduce novel usage scenarios to user-device interaction. Thus, application agnostic origination will have a multiplicative effect on the diversity in the meaning of context. For a better explanation, consider the example of a speech recognition interface for smartphones. Figure 4 shows the possible usage scenarios in rectangular boxes, interconnected by forms of context used in their classification.

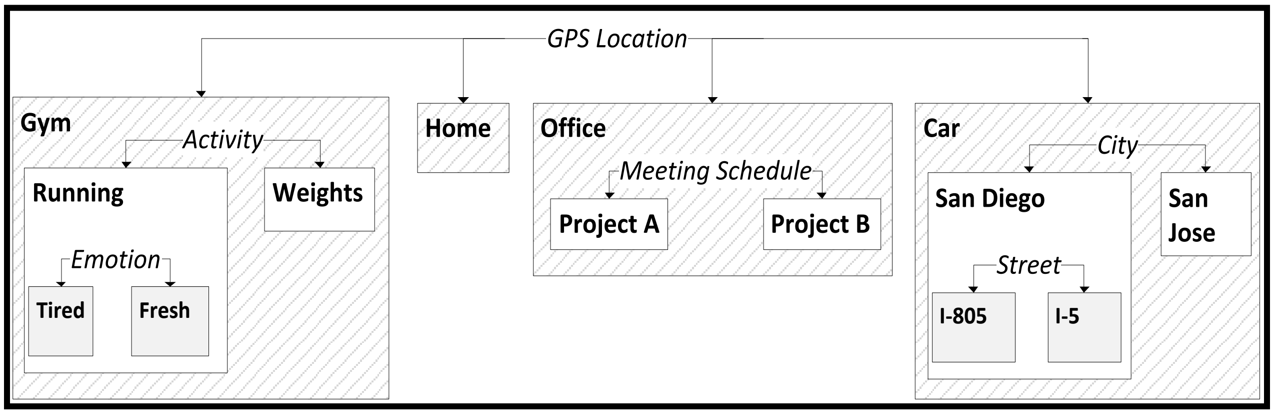


Figure 4: Example User Context for Smartphone Speech Recognition Interface

Immediately after its purchase the speech interface might be accessed exclusively at home and the user's vocabulary will be accordingly constricted. As the user acclimates with the device, its usage can extend into accessing email at office, navigation in the car or playing music at the gym and so on. Each of these usage scenarios will contribute a unique set of speech commands to the vocabulary. Between the car, gym, office and the home, a high level classification of speech can thus be developed with the user's GPS location. As user interaction and vocabulary within these usage scenarios rises, new classifications other than the GPS location will arise. At the gym the playlists selected might vary between the activity of running and weight training. Further, when running, music choices might be driven by emotion – manifested as motivational levels. At the office, the meeting schedule is a plausible classifier, as project specific files and emails are accessed. Within the car, the location at lower specificities, such as city (San Jose, San Diego) or the street name (I-5, I-805), can classify the destination for navigation. Even in this rather small example, up to 6 different meanings of context arise as vocabularies increase. As new applications are accessed, especially those whose usage extends to where speech interfaces are not used, such as in the cinema hall, or noisy public transport, this number will potentially multiply. Consequently, as the number of database dependent applications and their algorithmic complexities rise, user context with a diverse set of meanings will have to be leveraged for partition origination.

The existing approaches for developing and using context are specific to applications and usage scenarios [82, 92]. While these are all valid, they cannot satiate the required levels of diversity and will consequently originate only a limited set of partitions. Instead, a fundamental approach is required that bridges these and other existing approaches, and at the same time enables the exploration of novel meanings of context. This seems to be a tall challenge for computer systems, whereas the human brain seems to be adept at using context across myriad cognitive tasks and rising complexities [16]. Scientists believe that underlying this ability is a spatio-temporal model of the world developed over the human lifetime [93]. This resides in the hippocampus and helps classify cognitive inputs along appropriate spatio-temporal context, reducing the search space for memory retrieval and improving its efficiency.

We believe that a similar spatio-temporal approach in developing user context will allow for both scalability and application agnosticism in originating partitions. This is because, time and space are fundamental dimensions of the physical world, and they can together, in different forms, describe any usage scenario sufficiently. For example, activity classification between running and stationary can be deduced from the change of location over time. Even context drawn from emotion seems to be an internal tag given to a sequence of temporal–spatial information. For example, the emotion is of increased motivation, as the time spent inside the gym rises, or focused in an office meeting. Similarly, other meanings of context at different levels of specificity can also be derived from constituent temporal and spatial information. A spatio-temporal approach thus binds and underlies the existing high level approaches. Thus it provides the required fundamental approach that can potentially ensure scalable and application agnostic partition origination. The corresponding definition of context, as used in this research effort can then be cited as:

"Context is an accumulation of temporal, spatial information about the User, Device or Application that can be used for differentiating and classifying the input stream for a database dependent application".

To elucidate the use of this approach in originating partitions and introduce the reader to some terminology, we use the example of a smartphone based face recognition application. Two users working in the same office, one an engineer and the other a manager are considered. The partitions that can arise using the spatio-temporal approach as they encounter an increasing number of people are shown in Figure 5 (a) and (b) respectively.

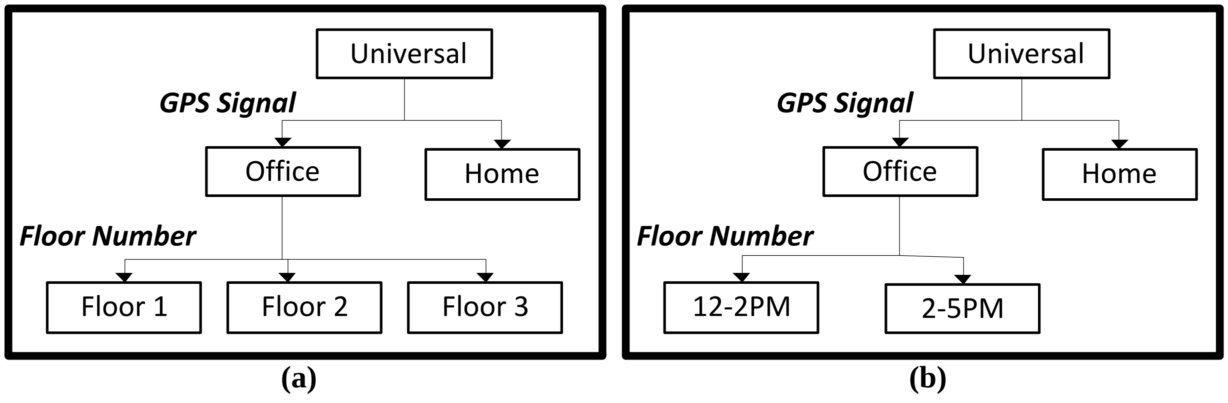


Figure 5: Example Spatio-Temporal hierarchy of partitions (a) Engineer (b) Manager

At the onset, when both users begin using the application, no classifications will have been made, and the initial set of faces will reside in a single partition. This can be considered to be a default partition that is agnostic of the user context and is thus referred to as the *Universal Partition*. As the number of people encountered rises, the size of the universal partition will become unwieldy and necessitate a further partition. Plausibly a partition for both users might exist between the GPS location at the office and the home. This will divide the universal partition's facial database into smaller, performance optimized set of children partitions-Home and Office. At this point, the Home and Office partition are the *terminal partitions* to this developing hierarchy of partitions whereas the Universal partition is a parent. As the user acclimatizes to the workplace, the interaction with colleagues will rise, eventually necessitating further partitioning even within the office. Here, differences can arise between the two users, as people met by a manager will plausibly repeat with project meetings. While for the engineer, his/her location between floors-Meeting Room on Floor 1, Human Resources on Floor 2 or the Lab on Floor 3 can classify the people met. The office partition will thus become a parent to a new set of terminal partitions. As algorithmic complexities continue to grow, new performance optimized terminal partitions will be added, resulting in an ever growing hierarchy of partitions that are inter-related by forms of time. We refer to this organization of partitions as the *Spatio-Temporal Hierarchy*.

The user context at any partition in the hierarchy is an accumulation of multiple contextual inputs. For example, when classified in Floor 2, the user's context is developed from the GPS signal and the department. This means that every set of terminal partitions represents a higher specificity of the user context compared to its parent partition. For example, in Figure 5(a), the first set of terminal partitions (Office and Home) classify user context in the overall universe, while the next set (Floor Numbers) classify it within the high level context of their parent i.e. the office. Thus, the same user context will pertain to multiple partitions at different levels of specificity in the hierarchy. However, the terminal partitions will be the most contextually specific and performance optimized options for real time usage. Overall as the algorithmic complexities rise, increasingly contextually specific partitions will be added to the hierarchy, personalizing the application ever closer to human behavior.

Mobile devices today offer numerous sources for extracting temporal, spatial contextual inputs. These can be obtained both from the hardware and even from the software, as mobile operating systems and internet provides a parallel world of activity. In device hardware this includes embedded sensors such as GPS Signals at various specificities, Wi-Fi ID's, and Compass, and in software the calendar schedule, website URL, and application info. Higher, more abstract forms of context such as emotion can also be inferred internal to the device, by analyzing and correlating information received from multiple sources. In other cases these higher forms are already available as embedded signals such as accelerometer (location over time) and heart rate sensors or from dedicated applications such as RF Tag readers. As the number of embedded sensors and the pervasiveness of mobile devices rises, going forward there will be no dearth of contextual information to leverage. This gives us confidence that the spatio-temporal hierarchy approach towards origination will ensure both scalability and application agnosticism.

4.2 Integration

Next, we investigate an approach for integrating existing partitions with new algorithmic complexities. The partitions that are eligible for integration are the ones whose context matches the user context that accompanies the algorithmic complexities. As previously discussed, the same user context can pertain to multiple partitions at different specificities in the spatio-temporal hierarchy. For example, in Figure 5 (a), the people encountered at Floor 2 also belong to the higher specificity of the Office and to the context agnostic-Universal partition. Based on the partitions that are chosen as candidates, we identify two possible approaches: *Isolated* and *Replicative* Integration.

In isolated integration, only the eligible terminal partition is chosen while all parent partitions at higher specificities are ignored. For the example spatio-temporal hierarchy in Figure 5(a), this is visualized in Figure 6.

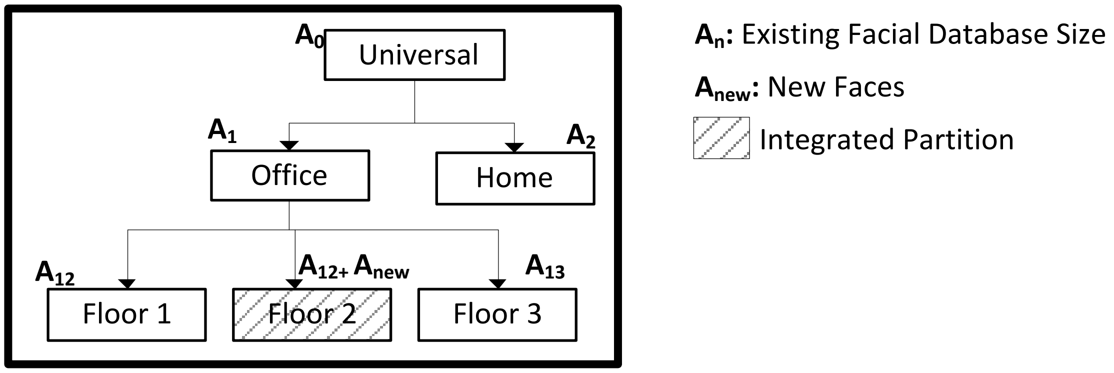


Figure 6: Isolated Integration Approach

The size of each partitions facial database (A_n) is printed in the top corner, and we consider that new faces equal to A_{new} are encountered in the Floor 2 context. In the isolated approach the new faces (A_{new}) will only be integrated into the Floor 2 partition, as it is the only pertinent terminal partition. This will accrue minimal overheads because irrespective of the number of partitions only one partition is integrated. Furthermore the increase in algorithmic complexity across the hierarchy is only $A_{new.}$ However, a key demerit is its inability to address integration inaccuracies. This is because the terminal partition is the only repository of new algorithmic complexities. If instead of the pertinent terminal partition an erroneous partition is integrated, there exists no intuitively identifiable backup partition. In Figure 6, consider that A_{new} is erroneously integrated into Floor 1 in place of Floor 2. In real time when the corresponding faces are received as inputs, Floor 2 will be unable to recognize them and it cannot be intuitively predicted that the faces are present in Floor 1 instead.

Alternatively, in replicative integration, new algorithmic complexities are integrated into all eligible partitions, irrespective of their specificity. This will redundantly replicate algorithmic complexities across the hierarchy. Because of replication the algorithmic complexity of each parent will be a super set of its children, and will present an intuitively identifiable backup to tackle integration inaccuracies. Figure 7 visualizes this approach.

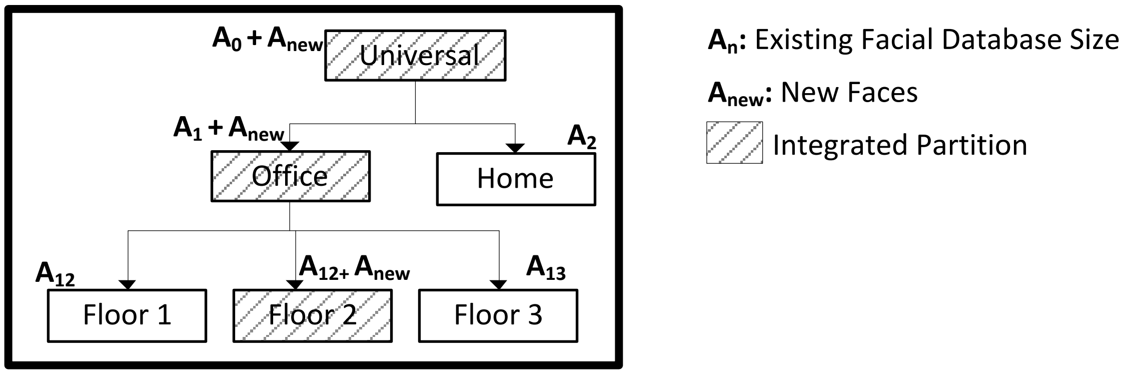


Figure 7: Replicative Integration Approach

As shown, apart from Floor 2 partition, the new set of facial inputs (A_{new}) will also be added to the Office and Universal partitions. Now, when Floor 2 is unable to decode the input, its parent partition-Office provides an intuitive backup. Further, if the complexity wasn't even integrated into the Office, its own parent- Universal Partition presents a backup. In this manner, the input can proceed higher in the hierarchy, and with every step higher in the hierarchy its probability of detection will increase, as the context becomes less specific. This will however also increase the performance penalty of the inaccuracy as the partitions keep getting bigger. However, in all probability the integration errors will be made between partitions with closely related contexts, thus the input will not travel very high into the hierarchy. Compared to isolated integration however, the increase in algorithmic complexities is equivalent to $3*A_{new.}$ Also, three distinct partitions must be integrated. This entails overheads three times larger than the isolated approach. As the depth of the hierarchy increases, a larger number of eligible partitions will emerge, entailing even higher overheads.

The two approaches-replicative and isolated, are at extremes from one another, and ideally an intermediate approach should be chosen. This will allow for the desired trade-off between the performance penalties of integration inaccuracies and the overheads of integration. For example, instead of integrating all the way up the hierarchy, only a sub set of eligible partitions might be chosen, to optimize the trade-off as per the user-application pairing. This however requires further investigation, and we leave its exploration open for future research efforts. Presently, we proceed with replicative integration because integration inaccuracies seem to be a greater concern for real time performance than slightly higher integration overheads.

4.3 Retirement

The insight that makes retirement possible is that the lifetimes of contextual patterns are finite, as a consequence of variations in user lifestyle. As patterns disappear, the corresponding algorithmic complexities will also become redundant, and can be removed from the spatio-temporal hierarchy. As replicative integration adds the same algorithmic complexity in multiple partitions, alternative approaches, differentiated by the partitions made eligible for removing complexities can be pursued.

A potential approach is to remove algorithmic complexities from all the partitions they are replicated into, as soon as the pattern is gauged to have disappeared. This amounts to removing all traces of the pattern from the spatio-temporal hierarchy in one step, we thus we refer to this approach as *Single Step Retirement*. For the face recognition application hierarchy visualized in Figure 6, this approach can be visualized as shown in Figure 8.

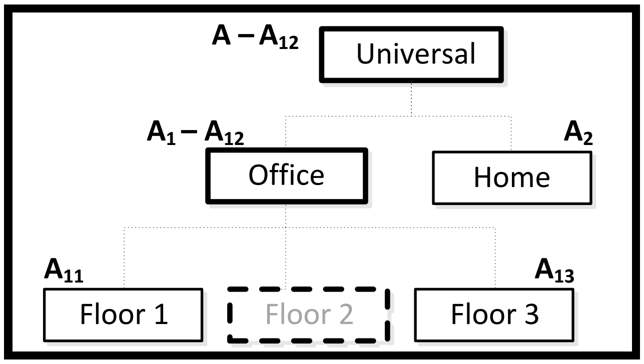


Figure 8: Single Step Retirement

As the user stops visiting Floor 2, the corresponding pattern will disappear, consequently the faces are removed Floor 2 partition (A_{12}), retiring it completely from the hierarchy. Alongside, the complexities are also removed from the Office and Universal partitions. Overall, this reduces the algorithmic complexities by 3* A_{12}, and architectural complexities by one partition. This is a highly optimistic approach which assumes that the contextual pattern will not reappear, as it maintains no backup for the corresponding algorithmic complexities. However, because of retirement inaccuracies, this will often not be the case. Consequently, single step retirement can result in large performance penalties.

To address this problem, another approach is to remove algorithmic complexities gradually over multiple steps, moving higher into the spatio-temporal hierarchy with every step, as the disappearance of a pattern becomes more certain. We refer to this as the approach of Multi-Step retirement, and it is visualized in Figure 9.

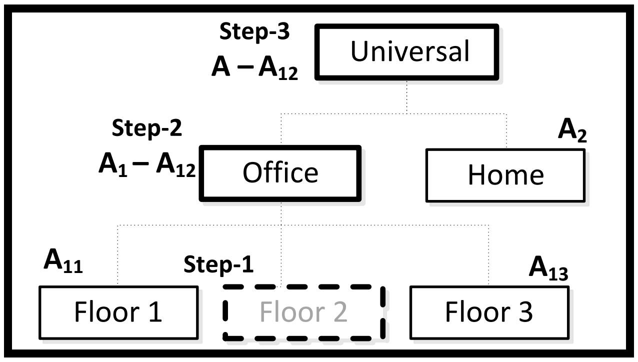


Figure 9: Multi Step Retirement

At the first step, the algorithmic complexities are removed only from Floor 2. As further time elapses and the user continues to not visit Floor 2, they are removed from the Office Partition in

Step 2, and finally from the Universal Partition in Step 3, after further time elapse. While the overall simplification attained will be the same as single step integration, it will is attained over three steps separated over time. This presents a more cautious approach, as backups of algorithmic complexities are maintained in spatio-temporal hierarchy. This means that if the corresponding pattern re-appears, the inputs can be decoded. For example, after step 1, the corresponding inputs can be processed by the Office database and after step 2 and the universal Partition can still be used. With every step higher into the hierarchy the penalties of inaccuracies will be larger as the partitions increase in size. However, as every step is coupled with passage of time, the probabilities of the inaccuracy will also decrease. We believe that incurring slightly higher architectural-algorithmic complexities with step wise retirement is reasonable considering its performance benefits, and thus choose it over single-step retirement.

This concludes our discussion on the malleable approach in contextual partitioning. The solutions we propose for the three initial challenges are: origination of partitions along a spatio-temporal hierarchy, replicative integration and step-wise retirement. We observe that as algorithmic complexities rise, contextual patterns will have to be identified at ever lower specificities for partition origination, thousands of candidate partitions will exist for integration and a proportional number of contextual patterns will have to be analyzed for retiring partitions. This means that the analysis for MCP will entail large overheads. Furthermore, as algorithmic complexities are replicated in multiple partitions in the spatio-temporal hierarchy, the overheads will scale in a super linear fashion with algorithmic complexities. As a result, even as the MCP approach has the potential of simplifying application performance, these overheads can overshadow its benefits as algorithmic complexities rise. It is thus critical to evaluate if the MCP approach can achieve the performance goals we set for ourselves in Figure 1. Towards this end, we require architectures that can accommodate these overheads, while continuing to offer performance benefits. Traditional parallel architectures however are limited in their scalability. Consequently, as the second major contribution of this research we investigate for an architectural framework that can implement the MCP approach and potentially allow for achieving our goals. We base our investigation on a recent area of research-Computational Dreaming that seeks to leverage dreaming in computers, particularly for improving architectural performance. In the next chapter, we discuss previous work in Computational Dreaming, and identify the challenges we face in the design of a Computational Dreaming architecture for the evaluation of the MCP approach

5 Computational Dreaming Architecture: Previous Work and Challenges

Scientists note the importance of dream sleep in simplifying awake operation of the human brain, and ensuring efficient performance despite growing cognitive complexities. Inspired from this, recent work has developed conceptual foundations of a novel area of research-Computational Dreaming [94]. Computational Dreaming has goals of leveraging dreaming in computers to improve architectural performance and understanding how novel forms of parallelism are related to intelligence. Unlike traditional parallel architectures, where overheads begin to dominate performance with PE numbers around 4-5, Computational Dreaming enables decoupling of architectural performance from its overheads, and can potentially allow for boundless scalabilities. MCP can be a candidate for Computational Dreaming, as dreaming can accommodate its scaling overheads without affecting real time performance, and in this way enable MCP to achieve the performance goals. Consequently, as the second major contribution in this research, a novel Computational Dreaming computer architecture is conceptualized for the MCP approach. We identify the following initial challenges as the DALI (Dream Architecture for Lateral Intelligence) is conceptualized for proof of concept:

- Implementation of MCP in a dream phase
- Selection of partitions in real time

Overheads in MCP accompany analysis that is required for originating, integrating and retiring partitions, and can be expected to scale along a super linear curve with algorithmic complexities (as discussed in the previous chapter). Thus, if performance goals are to be met, it is essential that this analysis be decoupled from the performance benefits of MCP. The phase of dreaming, as hypothesized [94], is devoid of user interaction, meaning that computation in this phase and its resultant overheads will have minimal impact on real time performance. Consequently, as the first challenge, we seek to implement the analysis for MCP in a novel dream phase of operation. In real time, the architecture can then leverage the benefits of MCP while incurring reduced overheads as a result of dreaming. To draw on the benefits of MCP, the architecture must be able to select a context specific partition for processing user inputs in real time or awake operation, which defines our second challenge. Here, we first discuss previous work in Computational Dreaming and the reasons why a Computational Dreaming architecture is a viable solution for MCP. Subsequently, we discuss the two initial challenges in further detail.

5.1 Previous Work

Dream sleep in the human brain is believed to be essential for continued efficiencies in memory formation, retrieval and general sanity, as even small periods of sleep deprivation appear to result in mental impairment [20, 21]. Previously, the few research efforts that have examined the implementation and benefits of dreaming in computer systems have been limited to the algorithmic level [5, 95]. Recent work [94] has proposed that dreaming is a consequence of physical limitations of the brain, and its implementation in computer organization can help achieve performance heretofore unachievable. Based on these observations, a novel area of research is proposed-Computational Dreaming. It is theorized that the human brain consists of multiple models of knowledge, each producing distinct outputs even for the same input. In the absence of dreaming, these models can diverge, making it impossible to select the correct set of outputs in real time. As a visualization of this hypothesis, in Figure 10 shows the brain, as a black box, (a) awake and (b) dreaming.

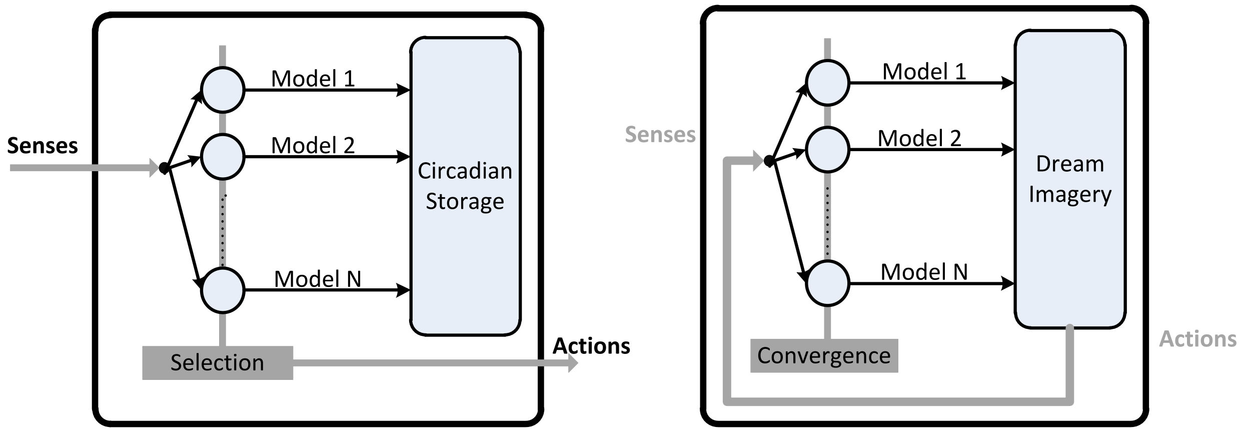


Figure 10: The Brain as a Black Box: (a) Awake and (b) Dreaming [10]

When the brain is awake and also alert, i.e. producing actions (outputs) from senses (inputs), multiple models produce outputs, which are recorded in circadian storage for later use; however the output from only one model is selected for daytime operation. While dreaming, the actions and senses are still present, however in much reduced state. The existing models now produce dream imagery, which is fed back for analysis. This allows them to converge with each other, and makes it possible for the brain to select the correct model output when awake. In computer architecture, these observations can be implemented to found novel architectural frameworks where multiple lateral processors, each modeling a distinct solution converge in independent parallelism in a dream phase, as a single processor is selected for real time operation. Independent parallelism when dreaming is essential, so that the awake/real time operation can continue to be simplified, even as the number of models - lateral processors continues to rise. This means that an architecture leveraging Computational Dreaming can continue to provide optimized performance with increasing complexity, unlike traditional methods of parallelism, where scalability is limited to 4-5 processing cores. Potential candidate applications that can leverage Computational Dreaming are those for which multiple potential solutions exist, however a single solution must be chosen in real time. These are characteristics held in common by database dependent applications, where user context drives the selection of a solution amongst multiple candidates. A Computational Dreaming architecture thus seems to be a viable solution for MCP to achieve the performance goals, and its development is pursued as our second major contribution.

5.2 Challenges

As was discussed in the previous chapter, the chief source of overheads in MCP is in support of the origination, retiring and integration of partitions. Using conventional techniques, these overheads can be expected to increase along a super linear curve with algorithmic complexities. The dream phase, as discussed in the previous sub section proceeds in the absence of user interaction, which means that computation implemented in this phase has the potential to have only positive impact on real time performance. The analysis for MCP uses only stored values from the user's prior experience, and it is thus feasible to implement it latently, while dreaming. The first challenge is to conceptualize a novel dream phase of operation for the implementation of MCP. Solving this will isolate real time operation from MCP’s overheads, making it possible to leverage only MCP’s benefits in real time. For this however, the architecture must be able to select context specific partitions for processing of real time application inputs, which becomes our second challenge. These challenges and the sub problems their investigation poses are now discussed in further detail.

5.2.1 Implementation

While dreaming is a latent phase that allows for the implementation of computationally complex tasks like MCP without affecting real time performance, it cannot take unscalable processing times for completion. This is because dreaming requires an absence of external inputs, which with mobile phones becoming constant human companions can be expected for only short durations, possibly coinciding with the users sleep cycle. This means that even as user interaction and resultant algorithmic complexities increase, the time available for dreaming will remain bounded and almost constant. These time limits for dreaming are consistent with the human brain, where through adulthood the sleep requirements remain stable around 6-8 hours., even as the brain continues to learn. Keeping these limitations in mind, for a dream phase to implement MCP, we identify the following initial problems:

- Developing a scalable algorithm for MCP
- Identifying memory elements for a condensed replay

For the implementation of MCP, the first step is developing an algorithm that analyzes application inputs and associated contextual inputs for MCP i.e. origination, integration and retiring partitions. As algorithmic complexities rise, this analysis will be implemented for an increasing set of partitions with numbers approaching thousands. However, as discussed above, only bounded times are available for dreaming. In order to comply with this constraint, the computational complexities of the MCP algorithm must scale with algorithmic complexities and partition counts. As discussed in chapter 2, traditional forms of parallelism are limited in the degrees of parallelism they can expose, as a result show close coupling between overheads and algorithmic-architectural complexities, and as a result do not provide viable solutions for developing an MCP algorithm. Thus, as the first problem, we investigate for novel forms of parallelism for developing a scalable MCP algorithm.

The MCP algorithm must analyze real time inputs, including the application inputs-images, audio stream, gestures and associated contextual inputs for origination, integration and retirement of partitions. Dreaming however will not have direct access to these real time inputs. To provide this access, memory elements are required for recording the real time inputs and subsequently replaying them when dreaming. However, as user-device interaction rises, an application can receive thousands of real time inputs. A replay can thus entail prohibitive times, making it impossible to complete within bounded time limits. At the same time, an incomplete replay will limit the personalization possible with MCP. The architecture must thus be able to condense this replay, so as to fit within the constrained time limits, while ensuring minimal loss in the fidelity with which real time inputs are represented. The identification of memory elements that can make a condensed replay possible thus forms the second problem we must address.

5.2.2 Selection

For the architecture to select partitions in real time, it must be able to schedule context specific partitions over suitable Processing Elements (PE's), and communicate received application inputs into the corresponding PE. The challenge of selection can thus be dubbed as the problem of designing a scheduler and an on chip communication framework for the Computational Dreaming architecture. While this is a canonical architectural problem, it is particularly unique if we are to leverage the benefits of the MCP approach. This is because, for the selection of partitions with user context, scheduling and communication decisions will have to draw upon external inputs, unlike traditional architectures where these decisions are based on system level parameters like pre-assigned arbitration priorities. Furthermore, as partitions originate and retire, the scheduler and on chip communication designs will have to be equally malleable to continue to offer optimized performance. The initial problems that the scheduler and on chip communication framework design must address are:

- Scalable real time performance
- Low overhead re-programming in the dream phase

Scheduling and communication overheads are key performance determinants for system on chip architectures [56]. They are even more critical for database dependent applications, because often, small time gaps will separate consecutive application inputs, such as words in natural speech or a sequence of gestures. Thus the overheads of scheduling and communications can add up to result in large performance penalties. At the same time, making an accurate selection can be computationally complex, as thousands of candidate partitions are added to the search space with rising algorithmic complexities. Thus, to ensure that the overheads of selection do not outweigh its benefits, we must investigate for scheduler and on chip communication framework designs that offer scalable performance against a rising number of partitions.

As partitions retire, and optimized alternatives originate in the dream phase, the partitions pertinent for real time usage will also change. In this scenario, to continue to select accurate and performance optimized partitions, scheduling and routing framework must be re-programmed accordingly. This is a unique problem, because while there has been recent interest in programmable scheduling and communication methodologies [96], traditionally the scheduling, communication frameworks have been static design time elements. Additionally, as algorithmic complexities rise, hundreds of partitions might be altered in the dream phase, and re-programming can entail large overheads. Thus, it does not seem reasonable to implement in the real time. While the dream phase provides an alternative, it is already burdened with the implementation of MCP, and does not offer unbounded processing times. Thus, it is critical that the scheduler and on chip communication framework designs allow for low overhead reorganization in the dream phase.

In this chapter, we discussed previous work in Computational Dreaming and the conceptual foundations for its implementation in computer architecture. We identified and analyzed two initial challenges in the design of a Computational Dreaming architecture for evaluating performance of the MCP approach. These are the implementation of MCP in a dream phase, and the selection of partitions in real time. In solution to these challenges, we conceptualize a novel architectural framework-the Dream Architecture for Lateral Intelligence (DALI). The next chapter discusses the solutions to these challenges and in this way the architectural framework of the DALI.

6 The Dream Architecture for Lateral Intelligence (DALI)

We now discuss the second major contribution of this research: the conceptualization of a Computational Dreaming architecture for the performance evaluation of MCP. We name this architecture the Dream Architecture for Lateral Intelligence (DALI), because the processors that implement the partitions operate laterally, or in perfect parallelism. Previously, we identified two challenges in its conceptualization, the implementation of MCP in a dream phase, and selection of partitions in real time. We first discuss the solutions developed for the two challenges and the resultant architectural components. Subsequently, we summarize the sequence of operations of these components in real time and in the dream phase of operation.

6.1 Implementation

The first challenge is the implementation of Malleable Contextual Partitioning (MCP) in a dream phase. For this, we identified two initial problems: developing a scalable MCP algorithm and identifying memory elements for a condensed replay. The discussion in this section is broken into the solutions for these problems.

Scalable MCP Algorithm

The first problem is developing a scalable MCP algorithm such that despite rising algorithmic complexities and partition counts its processing times can be constrained within bounds. Traditional forms of parallelism are subject to limited scalability [8], and are thus unsuitable for use in the MCP algorithm. We thus look for novel forms of parallelism to develop the MCP algorithm. A key insight is that even as the number of partitions approaches thousands, the decisions for their origination, integration and retirement will remain specific to each partition. This means that each partition can analyze for MCP laterally, in perfect parallelism and independence from the other partitions. Previous work in contextual partitioning [37] refers to this form of parallelism as Context Level Parallelism (CLP).

In CLP, partitions are the smallest unit of parallelism, and because they can process inputs laterally, there is no overhead of inter-partition communication or synchronization, unlike traditional methods. As a result, the overheads for CLP are negligible and not coupled with the number of partitions. Thus, CLP potentially promises boundless scalability against algorithmic-architectural complexities and qualifies as a candidate form of parallelism for the MCP algorithm. In leveraging CLP for the MCP algorithm, each partition will share a common input data stream - the replay, which is processed differently by each partition. As per Flynn's Taxonomy [44], this would qualify the DALI as a Multiple Instruction Single Data (MISD) architecture. This marks a break from traditional research that has focused on MIMD (TLP), or the SIMD (DLP) classification, as has been discussed in Chapter 2.

An initial question in developing the MCP algorithm is identifying an optimal scheme for allocating partitions to PE's. Previous work [37] that proposed CLP has recommended one to one allocation. However, initially we considered the option of multiplexing multiple partitions to a single PE. The two approaches: one to one and multiplexed are visualized in Figure 11 for the face recognition example discussed earlier in Figure 6.

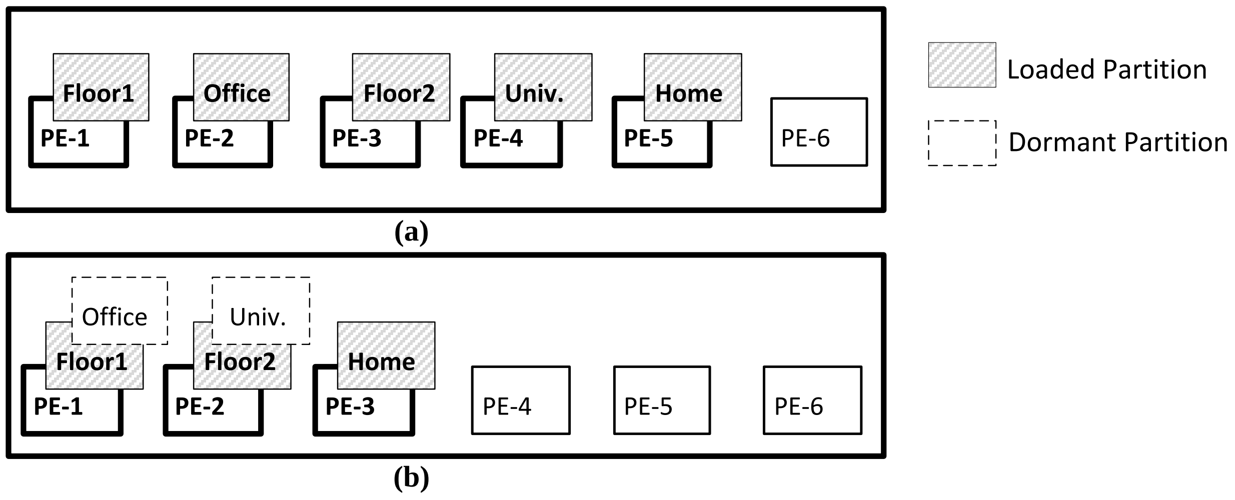


Figure 11: Processing Element-Partition Mappings: (a) One to One (b) Multiplexed

We consider an architecture with six PE's. In the one to one approach, the five partitions in our face recognition example (Universal, Home, Office, Floor1 and Floor2) require five distinct PE's. This means that all but one of the PE's (i.e. PE-6) are allocated to partitions. As each PE supports an individual partition, all partitions remain loaded onto respective PE's. Here "loaded" means that the facial database for the partition is available in the highest levels of the memory hierarchy including L1/L2 caches. Whereas, in the multiplexed approach each PE can carry more than one partition. In the figure, each PE is allocated a maximum of two partitions. Thus, the five partitions from the face recognition example allocate only three PE's, leaving three PE's completely un-allocated. However, for allocated PE’s, only one of the two partitions will be loaded into the PE memory, while the second partition lies dormant in the lower levels of the memory hierarchy, possibly even off chip memory, depending on the size of each partitions database.

The benefit of the multiplexed approach is that it exerts lower pressure on available architectural complexity. In our example, with the multiplexed approach there is room for seven more partitions-one on PE-3 and two each on PE-4, PE-5 and PE-6. Whereas, in one to one mapping, there is room for only one more partition. In case new partitions originate, for example, the Home partition is further split between two partitions: one for the apartment and another for the leasing office, the one to one approach will be forced into retiring an existing partition. On the other hand, the multiplexed approach can accommodate these without retirement, and still have room for five more partitions. In effect, with the one to one approach, partition retirement will be more frequent. As a result, the DALI will be more susceptible to retirement inaccuracies and their performance penalties, as have been discussed in Chapter 4.

However, the multiplexed approach can pose large real time overheads when a change in user context forces a swap between the loaded and the dormant partition. For example: as the user walks from Floor 2 to the office reception, PE-1 will load the Office partition replacing the Floor 2 partition. This swap will stall the PE on expensive memory cycles, even as inputs continue to be received and will thus severely impact real time performance. The one to one approach is devoid of this overhead, as the office partition is already loaded and ready to use. Another demerit with the multiplexed approach is the waste of CLP because multiple partitions contend for the same processing resources. Thus, all of them cannot execute the MCP algorithm in parallel. Contrarily, the one to one approach maintains the ideal levels of CLP, as each partition executes the MCP algorithm on a distinct PE. These disadvantages of the multiplexed approach dwarf its advantages, especially, for future architectures with thousands of PE’s, where pressure on architectural complexity will be a comparatively smaller concern. Nonetheless we recommend that future research consider a hybrid approach, where one to one mapping continues until all PE’s are allocated, following which multiplexing can begin. Presently however, we proceed with one to one mapping between partitions and PE’s for the MCP algorithm.

Having settled on a partition-PE mapping scheme, we next develop the MCP algorithm. Presently, we consider a homogeneous set of processing elements, and leave the exploration of heterogeneity for future work. The MCP approach, as discussed in Chapter 4 entails origination, integration and retirement of partitions. Presently, we cover only origination and integration in the MCP algorithm, and leave the implementation of retirement for future research efforts. Figure 12 shows the pseudo code of the developed MCP Algorithm.

```
For (Input in replay)
    Comparative Performance Analysis
    If (Terminal Partition)
        Compute Partition Performance Metric
        IF ( (Performance Metric > CPM) Then
            Origination Required
    Replay Analysis
    IF (Recorded Input Context == Context of Partition) Then
        Integrate Partition Database
        IF (Origination Required) Then
            Originated Partitions =Partition Origination Algorithm( )
            Allocate PE's to Originated Partitions
Retrain Application Algorithm
Generate and Output Feedback
```

Figure 12: Pseudo code for Malleable Contextual Partitioning (MCP) Algorithm

We discuss the pseudo code using the face recognition example from Figure 6. The MCP algorithm at a high level can be divided into two sets of analysis: comparative performance and replay. As dreaming begins, the partition-PE pairs receive a replay of real time inputs. Both the analyses are executed for every replayed input. The comparative performance analysis decides whether new partitions need to be originated. Significantly, this analysis need not be made for parent partitions, because optimal alternatives – their children already exist. For our face recognition example, the comparative performance analysis is run for Floor1, Floor2, Home partitions and skipped for the parent partitions-Univ., Office. The key insight is that to maintain constant performance, new partitions need to be originated only when an existing partitions violates the required real time performance. The required real time performance level is defined using the parameter-Context Performance Metric (CPM). The CPM can be defined directly as the required performance level in units of time, or indirectly, as the database size that ensures the required performance level. The latter is possible because for database dependent applications the database size is directly related to the application performance. For our face recognition example, the CPM can be defined as the face recognition time required by the user-device pairing. Or as the number of faces in the recognition database beyond which the required recognition time is not possible. The partitions performance is then computed and compared with the CPM, and if it violates the CPM, it is flagged as requiring origination. For the face recognition example, if the Home partition has more faces than the number defined as the CPM, it will be unable to maintain required recognition rate and will thus be flagged as requiring origination.

The replay analysis studies the real time inputs for partition integration and origination. Each real time input can be considered to consist of two sub-components: the application input and the accompanying contextual inputs. For example: for an input received by the face recognition application on Floor 2 of the office, the application input would be the face and the contextual inputs would be the GPS Location: Office and the Floor #: Floor 2. The context of each input is then compared with that of the partition, and if they match at any level of specificity, the application input is integrated into the partitions database. For our example input, the context of the input described above will match with Floor2, Office and the context agnostic Univ. partition. Accordingly the received face will be integrated into these three partitions.

Next, for the partitions that were flagged as requiring origination, the partition origination algorithm is executed. This algorithm identifies contextual inputs in the replay stream to divide the parent partition database between new children partitions. In current research we do not develop a partition origination algorithm, leaving it as a future research target. Nonetheless we are confident that this algorithm will not bottleneck the MCP algorithm. The reason is that a partition need not search through the entire replay stream; instead it can limit its search space to contextual inputs falling within its own context**.** For example: in order to divide the Home partition, the contextual patterns from when the user is at the Office can be ignored. This reduces the search space of the partition origination algorithm and ensures scalability despite rising partitions numbers. Finally, the originated partitions are allocated to the available processing elements. After all inputs in the replay have been analyzed, each partition-PE re-trains its application algorithm to account for any changes in its knowledge database. Finally, the partition-PE's generate and output feedback, to re-program the real time partition selection methodology of the DALI, as will be discussed in the next sub-section. In the face recognition example, say if the Home partition was further divided between the apartment and the leasing office. Then, the selection methodology will have to be re-programmed to select from the children in real time, instead of the parent. This would ensure that the DALI continues to select optimized and accurate partitions in real time. This marks the end of the MCP algorithm.

Memory Elements for Condensed Replay

The next problem is the identification of memory elements that are suitable for a condensed replay. A replay is required because dreaming is a latent phase, and does not have access to real time inputs, thus these must be stored in real time and later replayed to serve as inputs to the MCP algorithm. Initially, we considered using the local memory of the partition-PE pair that is selected for a real time input to also store it for the subsequent replay. This is an intuitive solution and does not introduce frivolous communication overheads in storing for the replay. However, in replaying the stored inputs, large communication overheads will be involved. This is because the real time inputs will be stored in a distributed fashion across the PE's, and for each partition to analyze all of the real time inputs, the PE's will have to communicate with each other. This approach is visualized in Figure 13 (a). If N PE's are used in real time, then in the dream phase, at any time N nodes might be arbitrating for the communication framework to exchange recordings with rest of the N-1 nodes. Consequently, the replay overheads will scale as a function of the number of partitions, resulting in the replay taking ever longer and defying the aim of its condensation.

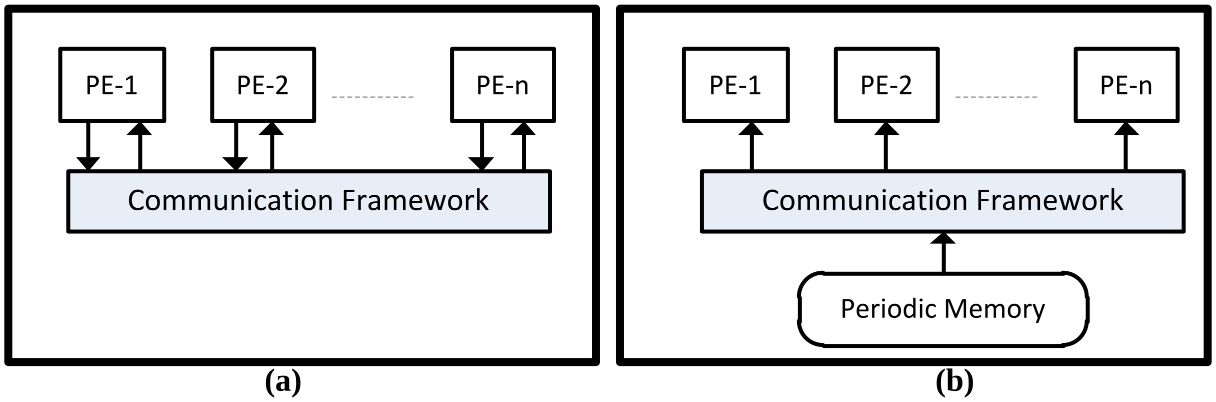


Figure 13: Memory Elements for Replay (a) Local Storage (b) Common Storage

As an alternative, a dedicated memory element, common to all processing elements, can be used for storing real time inputs. Its benefit is that a replay will entail minimal overheads, because the common memory element can broadcast onto all processing elements. We refer to this memory element as the Periodic Memory, because it will be periodically emptied in the dream phase, as it replays the recorded inputs. Figure 13(b) visualizes the replay with the periodic memory. As shown, a single node i.e. the periodic memory arbitrates for the communication framework to broadcast the recorded real time inputs to all PE's. Thus, the communication overheads will remain almost constant, even as the number of partitions rises, resulting in a much more condensed replay in comparison to the previous approach. A drawback of this approach is comparatively higher real time communication overheads, because inputs must now be communicated to both the selected PE and the periodic memory. This will however affect the performance minimally, because the periodic memory plays no active role in real time, and the storage can proceed in parallel to the input processing. Thus, the periodic memory element approach seems to be a more viable solution for the DALI.

6.2 Selection

The second major challenge is the selection of context specific partitions in real time. For this, we require a scheduling and an on chip communication framework that offers scalable real time performance and allows for low overhead re-programming in the dream phase. We first discuss the solution developed for the scheduler and subsequently discuss the choice for the communication framework.

Scheduler

Traditionally, schedulers in mobile architectures execute as software threads and compete for architectural resources with the other system threads. However, because of their importance in coordinating system activity, scheduler threads are given higher arbitration priorities and can pre-empt other threads. Schedulers thus often interrupt application operation and consequently their overheads impact system performance strongly. Such interruptions are not feasible for the DALI, if it is to achieve constant real time performance. At the same time, scheduling needs to be prioritized because the user's context can change frequently and application inputs will often be received with small time gaps. Considering this, a dedicated architectural space for scheduling in the DALI seems to be the most viable solution. This solution also seems to be feasible considering that with multi-core scaling; the number of PE's on a chip will not be a constraint in future architectures. We thus dedicate a processing element for scheduling in the DALI. It is referred to as the *Context Activation Processor (CAP)*. In current research we conceptualize CAP to be a general purpose processor; however we recommend that future efforts investigate its implementation as specialized hardware.

Traditionally, for schedulers, a critical design parameter is the strategy for multiplexing PE's between software workloads. The DALI however implements one to one mapping between PE's and partitions, thus no multiplexing strategies are required for CAP. Instead the key scheduling concern is the selection of a partition that is accurate to user context and optimized in performance, without spending unscalable times. Currently, we only conceptualize and outline a potential CAP scheduling algorithm, to prove that scheduling with the CAP will not bottleneck real time performance. The proposed CAP scheduling algorithm uses a scheduling database that maps partitions to the contextual input values used in their classification. Figure 14 visualizes this scheduling database for the face recognition example previously discussed in Figure 6. Apart from the partitions previously discussed in Figure 6, we generalize the spatio-temporal hierarchy to consist of N+1 partitions, where the Nth and N+1th partition are classified with the contextual input-X.

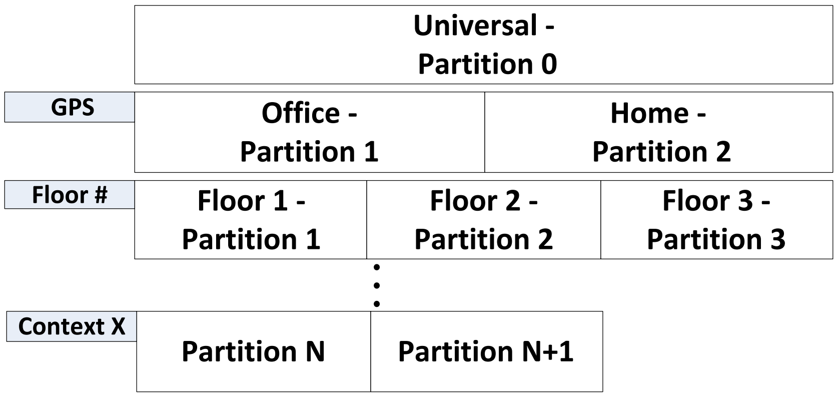


Figure 14: CAP Predictive Scheduling Database

The scheduling database tags partitions with the corresponding values of contextual inputs such as GPS, and Floor #, except the universal partition which carries no tag as it is context agnostic. The benefit in using a software database is that as new partitions originate and others retired, re-programming of the scheduler will require only an addition/removal of corresponding database entries. This will entail minimal overheads, ensuring that re-programming can fit into the bounded times for dreaming, even against a rising number of partitions. A limitation of database centric algorithms is their scalability, because the overheads of maintaining and accessing it increase with its size. This however, will not be a limitation for CAP, if the following two observations are leveraged: First, context at lower specificities will follow that at higher specificities in time. Second, context at higher specificities will stay relatively constant over long periods of time. Combined, these two observations can keep the search space for scheduling constricted to a subset of predicted partitions. This predictive approach to scheduling is visualized in Figures 15 (a) and (b).

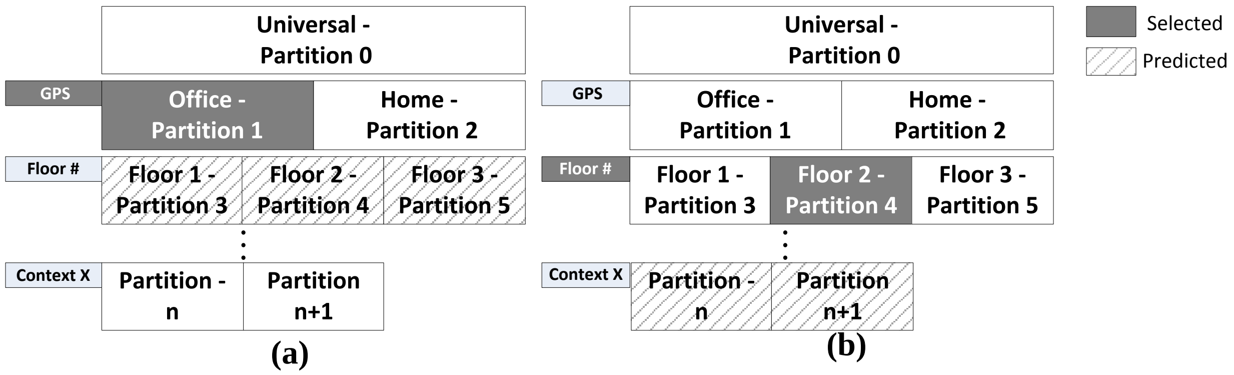


Figure 15: Predictive Scheduling with CAP

As the user parks into the office lot, the GPS signal will change to the office location. At this point one out of only two partitions (Office and Home) has to be chosen. Also at this point, the algorithm can predict with high certainty that context at the next level of specificity will correspond to its children partitions-Floor 1, Floor 2 and Floor 3. This will constrict the search space of the next step to only three out of six partitions, and with more analysis of past behavior, the algorithm can even pre-select the partition gauged to be the most probable–possibly the Floor with user's office. The state of the algorithm with respect to the database in this state is shown in Figure 15 (a). Next, as the user enters a Floor, the corresponding partition can be selected, and the selection of its children predicted, as shown in Figure 15 (b). This process of predictive selection will trickle through the database, optimizing the search space of the scheduling algorithm at each step. Further, for 8-10 hours, the user will remain in the office, and the algorithms search space will also remain constricted to the three Floor partitions. Consequently, even this optimized predictive selection will have to repeat only over a subset of the database, further lowering its overheads. Predictive scheduling will thus keep database accesses optimized, and make possible scalable real time performance even as the number of partitions continues to increase.

A notable point that we have not covered is scheduling in case of integration and retirement inaccuracies. To address this, the scheduling algorithm in CAP must be able to detect inaccuracies and schedule alternative partitions accordingly. The discussion on integration and retirement inaccuracies in Chapter 4 provides a solid foundation to address this. This presents an interesting challenge for future research efforts. However, in present work we consider only accurate operation for the DALI.

On Chip Communication

In older System-On-Chip (SoC) designs, communication frameworks were developed using hierarchical bus or crossbar interconnects. However, as has been discussed in Chapter 2, these offer limited scalabilities. Additionally, because their design is almost entirely hardware based, re-programming even where possible is accompanied by large overheads. Recent SoC designs incorporate Network-on-Chips (NoC) as alternatives to these traditional designs. The benefits in using NoC based solutions are higher scalabilities and flexibility in removal / addition of communicating nodes, which means low overhead re-programming as partitions retire/originate.

Consequently, NoC seems to be a viable solution for the DALI's framework. The NoC in the DALI is referred to as the *Context Sensitive Network* (CSN), because the routing decisions will depend upon user context. The routing tables in CSN will carry entries that pair all existing partitions with the allocated processing elements. In real time, the CSN will use the partition selected by the Context Activation Processor (CAP) to select the corresponding processing element, and route the application input into it. Much like the CAP, the re-programming of CSN will be limited to an update of these routing tables, and will consequently entail minimal overheads.

This concludes the discussion on our solutions to the two major challenges for the conceptualization of the DALI. In the next section we summarize the architectural framework of the DALI and its operation in real time and the dream phase.

6.3 Sequence of Operations

We have previously discussed operation of each component individually. Here, to understand their inter-operation we discuss the sequence of operations for the DALI in the real time and the dream phase. In the previous sections, we identified 4 key architectural components for the DALI, these are: Periodic Memory, partition-PE pairs, Context Activation Processor (CAP) and the Context Sensitive Network (CSN). The resultant architecture of the DALI can be visualized as shown in Figure 16.

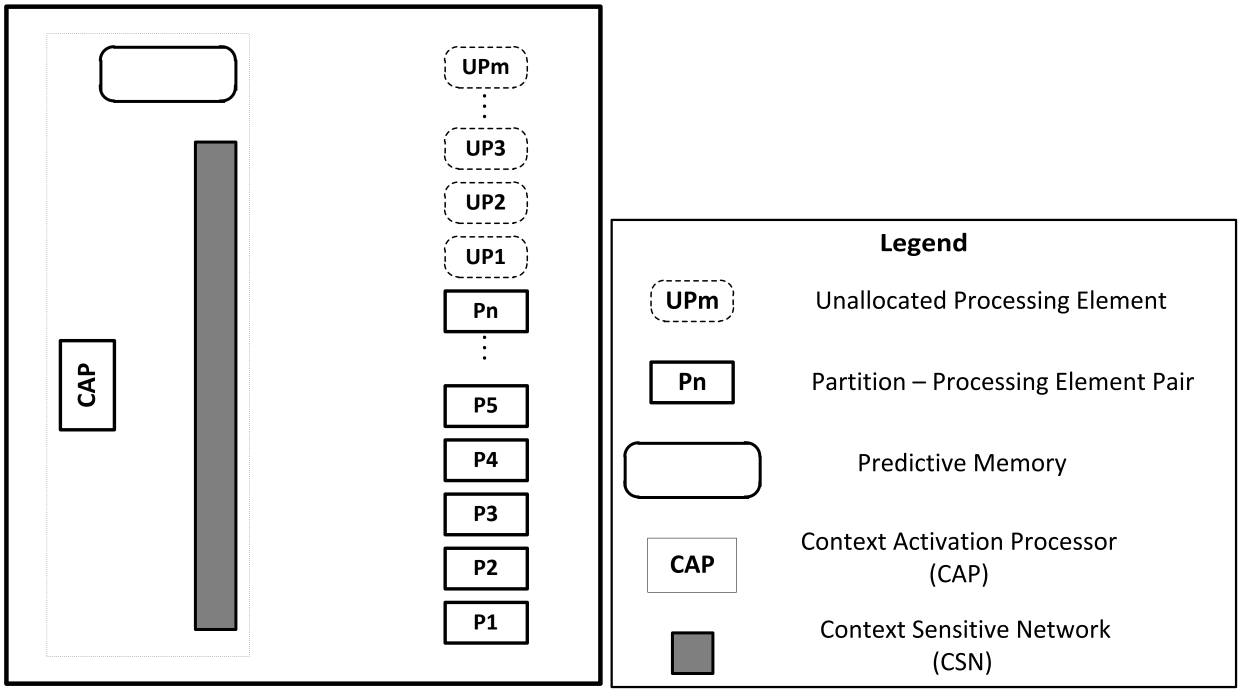

UPm
UP3
UP2
UP1
Pn
P5
P4
P3
P2
P1
CAP
Legend
UPm
Unallocated Processing Element
Pn
Partition – Processing Element Pair
Predictive Memory
CAP
Context Activation Processor (CAP)
Context Sensitive Network (CSN)

Figure 16: The Dream Architecture for Lateral Intelligence (DALI)

This represents an architectural state where N partition have been originated, and allocated to an equal number of PE's, while M PE's remain un-allocated. Correspondingly, CAP and CSN will carry scheduling, routing database entries for all N partitions. First, we discuss the DALI's dream phase operation, with Figure 17 visualizing the same.

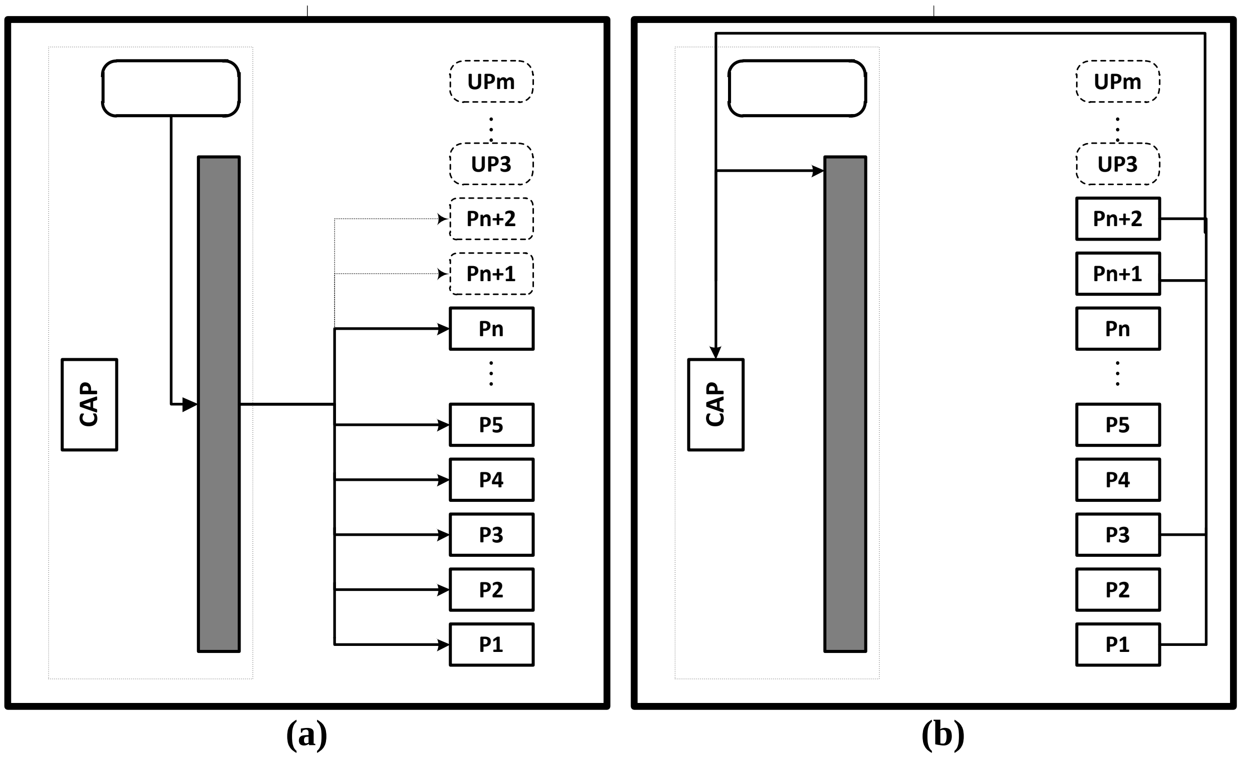


Figure 17: The DALI in Dream Phase (a) MCP Implementation, (b) Reprogramming CAP, CSN

Figure 17 (a) depicts the implementation of MCP in the dream phase. In the beginning, the periodic memory replays user inputs, including both application and contextual inputs, which were stored in real time. Each partition-PE pair then executes the MCPA, to integrating, originating partitions, and allocating any originated partitions to PE's. In the Figure, Pn+1 and Pn+2 are the two new allocated processing elements. Following their allocation, even the new partition-PE pairs i.e. P_{n+1} and P_{n+2} become recipients of the replayed inputs, as is represented by a dotted input line. The MISD nature of the MCP algorithm is easy to observe here, as a single stream of input – the replay is processed by each partition, and each partition makes differing decisions based on its context and performance requirements. When the partition-PE pairs finish processing all of the replayed inputs, the MCP implementation concludes. Following this the CAP-CSN must be reorganized to account for the changes in partition-PE pair mappings. Figure 17 (b) visualizes this reorganization. In this, the partition-PE pair's feedback into the CAP and CSN to update the scheduling and routing databases. The feedback consists of contextual inputs that have been identified and used to originate the corresponding partition, and ID's for the PE's they are allocated to. However, as these mappings are altered only for newly originated partitions, only a subset of partitions might need to feedback. The CAP and CSN then update the scheduling and routing databases, and add/change entries received in the feedback. The reorganization concludes when the CAP, CSN finish updating respective databases.

Overall, the dream phase can be sub divided into two constituent phases: the MISD algorithm for MCP and CAP-CSN reorganization. The reorganization of CAP-CSN entails a limited number of low overhead database updates, and will take negligible times, especially in comparison to the MISD algorithm for MCP, which is required to operate over thousands of partition-PE pairs. As a result, it is safe to classify the dream phase as an MISD phase of operation. With the conclusion of the dream phase, the DALI's architectural framework is prepared to offer optimized performance in the real time operation that follows. Figure 18 then visualizes the subsequent real time operation of the DALI.

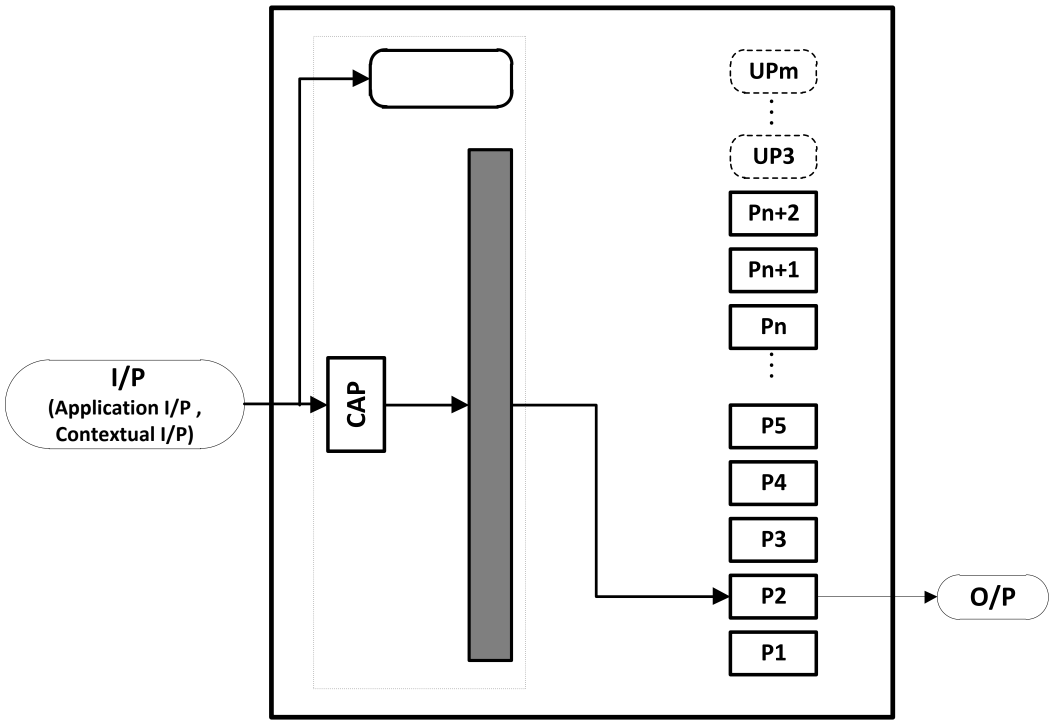


Figure 18: The DALI in Real Time

Following the dream phase, n+2 partition-PE pairs exist on the DALI's architectural framework, and the scheduling database in CAP, routing tables in CSN are configured to select one partition-PE pair specific to the users' context. As the user interacts with the device, inputs (contextual inputs and application inputs) are received. These are stored in the periodic memory, representing experiences the user had but did not have time to fully process, while the CAP uses the contextual inputs for scheduling a context specific partition-PE pair. The scheduled partition alongside the application input are then communicated to the CSN, which routes the application input to the corresponding processing element (P2 in this case). The selected partition-PE pair then produces the final output, and these steps repeat for all user inputs.

Significantly, as per Flynn's Taxonomy, the DALI's real time operation would correspond to the Single Instruction Single Data (SISD) classification. This is because a single PE is used to process the user input. Furthermore, even the single PE, executes only a subset of the algorithmic complexities, corresponding to the allocated partition. On the other hand, in the dream phase all allocated PE operate in MISD parallelism, which means greater architectural activity and proportionally, larger power consumption compared to real time. This is consistent with the human brain, which is observed to be highly active during dream sleep, consuming the same or even higher energy than when awake [97]. Scientists believe that this activity corresponds to memory consolidation in the human brain, which is required for efficient and faster memory retrieval, as discussed in the background chapter. Similarly, heightened architectural activity and greater power consumption in the dream phase for the DALI is necessary to make the contextually simplified SISD real time operation possible. However, the dream phase does not require user interaction, it can thus safely be implemented as the mobile device is plugged in for battery charging, meaning that heightened power consumption will not impact user experience.

In this chapter, we conceptualized the DALI as a novel architectural framework. The DALI implements MCP in a dream phase, which in turn enables contextually simplified real time operation. The goal of MCP and the DALI is to achieve constant real time performance against rising algorithmic and architectural complexities. In order to examine if the DALI is successful in achieving this goal, we setup an experimental investigation using speech recognition as an example database dependent application. In the next chapter we discuss the setup of an experimental platform for this investigation.

7 Experimental Platform

We setup an experimental platform to examine if the DALI can achieve the performance goals defined in Figure 1, with example application to speech recognition. Speech recognition is a representative database dependent application and prominent tool in developing natural human computer interfaces. However, despite decades of research, there still exists a gulf between the performance of current solutions and that of the brain [32]. Speech recognition has also previously been shown to be amenable to contextual partitioning [37], thus it presents an interesting candidate for early implementation of the DALI. The experimental goal is to examine if the DALI can maintain speech recognition in constant performance time as vocabulary sizes and partition-PE pairs increase. Simulators for DALI did not exist prior to this thesis, thus we develop a DALI simulator as a contribution of this work. Here we first discuss the setup of speech recognition for DALI, and subsequently the DALI Simulator.

7.1 Speech Recognition Setup

We created a vocabulary database personalized to a hypothetical user personality based on the plausible lifestyle of the user and defined a set of contextual partitions in the vocabulary database. Secondly, we chose and set up a software suite for training speech recognizers over the vocabulary database.

7.1.1 Vocabulary Database and Partition Definition

Research efforts in speech recognition have established and utilized numerous vocabulary databases, including popular options such as the Wall Street Journal Corpus [98] and Alpha Numeric Corpus (http://www.speech.cs.cmu.edu/databases/an4/). These come with pre-recorded audio streams corresponding to the contents of the vocabulary database. As a result, they have proven to be ideal for general purpose, speaker independent research efforts, as they cut down on experiment setup times. However, we target single user, personal mobile computing devices, where research shows greater recognition accuracies with speaker dependent training [99]. Additionally, to make partitioning possible, the vocabulary database must be personalized to user lifestyle and behavior. As a result, we develop a speaker dependent, contextually tagged vocabulary database specific to our research effort. This is accomplished by personalizing the vocabulary database to a hypothetical user personality.

The hypothetical user is a Virginia Tech student, who is also an avid follower of Major League Baseball (MLB). We identify a set of 7 possible contextual inputs that can be drawn for the hypothetical user. This set represents plausible contextual situations a user can encounter, and contextual sources that are easily accessible even in present day mobile phones. These are:

- Application Type
- GPS Location
- Webpage URL
- Time of Day
- Webpage Field
- Address Field
- Temporal History

Using these contextual inputs we define a set of 16 possible contextual partitions in the user's vocabulary database. Figure 19 shows the 16 possible partitions, and depicts their organization in the spatio-temporal contextual hierarchy. The partition name and its abbreviated form are shown in rectangular boxes, with the contextual input used in their origination printed above them. The abbreviated names for each partition are constructed using the following Naming Convention: *The names of the children partitions contain the parents name as the first component, thus a parents name will always be a subset of its child.* Example: MAPS_B is a child of MAPS and MLB_P_T is a child of MLB_P in Figure 19. This makes the identification of terminal partitions easier, as the ones whose names are not a subset of any other partitions name. This is because the terminal partitions are not parents to another partition, as discussed in Chapter 4. This convention is not followed for the Universal Partition (Univ.), to mark that it is context agnostic and a default partition. The only scenario where Univ. can be a terminal partition is when no other partition exists.

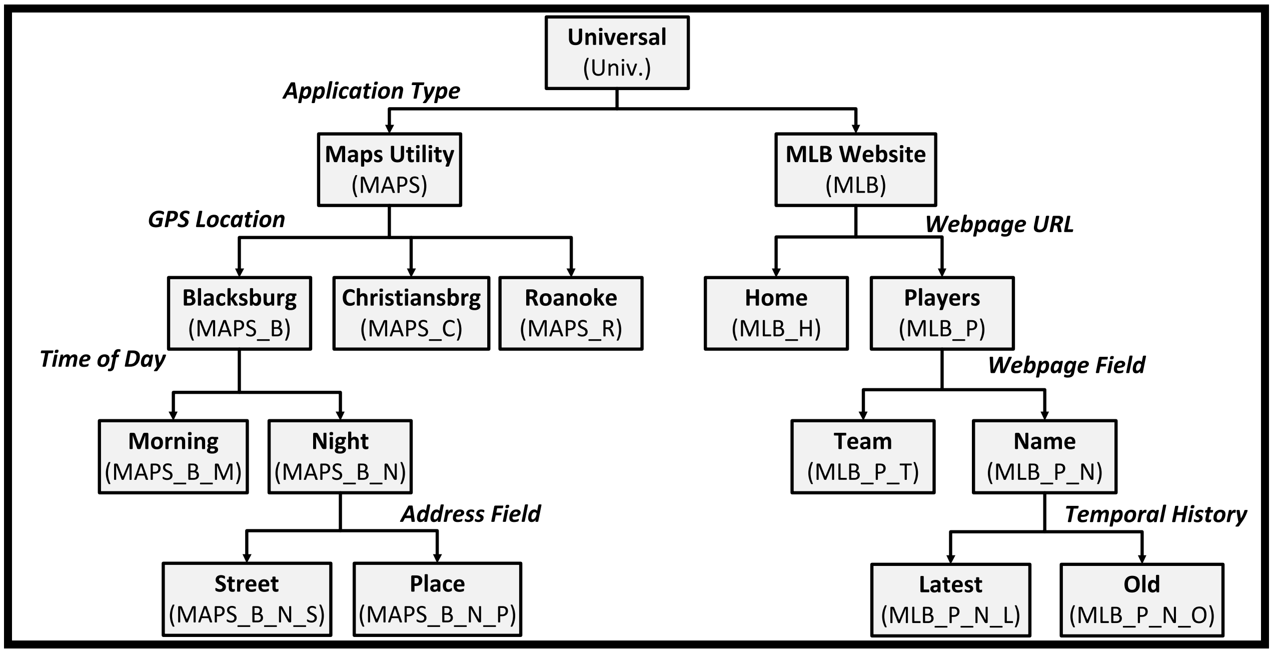


Figure 19: Spatio-Temporal Hierarchy for Example User

The default, context agnostic Universal Partition (Univ.) forms the starting point of the hierarchy and is positioned at the highest level of specificity, visualized by its position at the top of Figure 19. To understand the origin of other partitions, we discuss the meaning of contextual inputs used in their origination:

Application Type: The application being accessed is a strong marker of the spatial location of the user within the mobile operating system. Furthermore, applications with distinct purposes will often have markedly different menu items or command structures for their control. As a result, the vocabularies used in accessing them will also differ, allowing for their partitioning. In accordance with the interests of our hypothetical user personality, we identify two applications that the user can access. These are a mapping utility for navigating in and around Virginia Tech, and the MLB Website for tracking game results, player information, or participating in fantasy online leagues. This forms the basis of the first set of non-default partitions, MAPS and MLB respectively.

GPS Location: The vocabulary used with the Mapping Utility will be specific to the user location, as the user would want to navigate in and around a specific town/city. GPS is the most widely used indicator of user location, it can thus be used to partition the vocabulary contents of the Mapping Utility. We use three GPS locations specific to the area around Virginia Tech i.e.

Christiansburg (MAPS_C), Roanoke (MAPS_R) and Blacksburg (MAPS_B) to develop these partitions.

Webpage URL: Similar to spatial variations of speech in the mapping utility, the webpage URL can be used to classify the spatial location of the user behavior when exploring the MLB website. The differences in the vocabulary contents exercised by the user arise because of varying addressable content on each webpage. We use two possible values of the Webpage URL: the Players Page (MLB_P) and the Home page (MLB_H).

Time of Day: Time in different specificities i.e. hours, phases of day, weeks is an easily obtainable contextual input and a major determinant of user schedule. For the example personality it is highly plausible that while the user is in Blacksburg, he will access the of Virginia Tech campus over the Morning as the university is open and other places around the town of Blacksburg over the Evening-Night. Based on this, we use the phase of day to partition the vocabulary contents of the partition corresponding to Blacksburg (MAPS_B) in the mapping utility. This gives use the two partitions of Morning (MAPS_B_M) and Evening-Night (MAPS_B_N).

Webpage Field: Many webpages elicit explicit user feedback by requiring the fill up of online forms. For Example: The Players Page on the MLB website requires the user to fill up the Player and Team Names, to refine the search for the player statistics. This represents a finer specificity of spatial classification within the Players Webpage Context (MLB_P), as the user will use distinct vocabulary to fill in these fields. Thus, we use this signal to partition the vocabulary contents of the Players Webpage into Team (MLB_P_T) and Player Names (MLB_P_N) respectively.

Address Field: Similar to the webpage field discussed above, the mapping utilities generally have separate fields for filling in the Place Name and the Street Address. Using the cursor / mouse pointer the spatial location of the user's cursor can be identified, and used to partition the vocabulary used with the mapping utility interface. For the example personality we partition the vocabulary contents of the Night/Evening phase in Blacksburg (MAPS_B_N) into Place Names (MAPS_B_N_P) and Street Names (MAPS_B_N_S).

Temporal History: The set of Players that the user is actively interested in varies over time, as a result of retirements, injuries, trades between teams and popularity. This means, that with time the probability of usage of some names will be higher than the others, as some names fall out of usage, and will be replaced by new names. As a result, the vocabulary pertaining to player names can plausibly be partitioned into Latest (MLB_P_N_L) and Older (MLB_P_N_O) partitions. However, unlike earlier partitions, only one of the partitions-MLB_P_N_L might be used actively, with the other forms the backup, in case an older name, not a part of the latest partition is used. As new names are added to the vocabulary, the temporally older names can be moved out into MLB_P_N_O over the dream phase.

In current investigation we focus on isolated word recognition, and add words that can be used in each contextual scenario to the vocabulary database of the corresponding partition. This includes landmarks, restaurants, street names, apartment complexes, and other building names specific to the area surrounding Virginia tech, for the mapping utility and navigational commands specific to the MLB website, player names, and team names. for the MLB website navigation. Player Names in particular were adjusted to account for the geographically closest team (Washington Nationals) and all the players on their roster were added to the vocabulary. Each word is tagged with the contextual inputs that correspond to it from the spatio-temporal contextual hierarchy shown in Figure 19. For example: every word in the MLB_P_T partition is tagged with Application Type: MLB, Webpage URL: Players, Webpage Field: Teams. Overall, the vocabulary database consists of 300 words, and the sizes of each partition are listed in Table 3.

Partition Name	**# Words**	**Partition Name**	**# Words**
Univ.	300	MLB_P_T	30
MLB	160	MLB_P_N	60
MAPS	140	MAPS_B_M	25
MAPS_B	70	MAPS_B_N	45
MAPS_R	36	MLB_P_N_L	35
MAPS_C	34	MLB_P_N_O	25
MLB_H	70	MAPS_B_N_P	30
MLB_P	90	MAPS_B_N_S	15

Table 3: Partition Sizes in the Experimental Vocabulary Database

As can be observed in the table, the vocabulary size of each parent partition equals the sum of its children. This is because we aim to implement replicative integration in our experiments. While a scenario may arise where parent partition sizes are not necessarily equal, but are instead super sets of children. For example: when partitions originate, the contents of the parent might not be cleanly separated into child partitions, as some vocabulary contents might be replicated across the child partitions. This is however a corner case scenario, which can be taken up in subsequent research efforts. Next, we discuss the choice and setup of a software suite for training speech recognizers over this vocabulary database.

7.1.2 Speech Recognition Suite Selection and Setup

A host of speech recognition suites, both from the proprietary and the open source domains are available. Some popular proprietary solutions are the Hidden Markov Model Toolkit (HTK) and the Dragon’s Naturally Speaking, while Julius and Carnegie Mellon Universities (CMU) Sphinx are some open source suites. Between open source and proprietary solutions, the former is more advantageous, because of greater transparency, in that it allows altering configuration parameters at a much lower granularity, and allows for custom solutions that are in line with research goals. Amongst the open source candidates, we chose to use the CMU SPHINX recognition toolkit. The reason for this is its popularity as a recognition suite in previous research efforts [43, 100], and an active development and support base. Additionally, it offers proficiency across multiple languages, especially English and an ecosystem of other utilities such as language model generators, example speech corpuses, and active patch releases. The CMU Sphinx toolkit consists of SphinxTrain, which is an acoustic model training tool, and three sets of libraries for developing speech decoders, these are:

- Sphinx 4: a Java based speech decoder/recognizer,
- Sphinx 3: a C based speech decoder/recognizer
- PocketSphinx: Lightweight C based decoder for embedded systems

Amongst the three options, we chose PocketSphinx libraries for developing acoustic models and decoders as our focus is on mobile computers which lie in the larger domain of embedded systems. The Sphinx Train is then used to train acoustic models for our partitions, while the speech decoders are developed in Pocket Sphinx to use the trained acoustic models.

To train the speech acoustic models with Sphinx, we followed the steps indicated by the CMU Sphinx website (http://cmusphinx.sourceforge.net/wiki/). Some key files in the training of acoustic models, their purpose and setup for our experiments is as follows:

- Dictionary Files
- Unit List Files
- Language Model
- Test/Train File ID's and Transcription

The dictionary files list each word in the vocabulary database in terms of constituent phonetic units. Previous research [37] in contextually partitioned speech recognition has shown that phonemes provide sufficient accuracies and lower training times than other units for partitioned vocabularies. Consequently, we use phonemes as the standard phonetic units for all our experiments. The pronunciation of each word is obtained from the CMU's knowledge base utility (http://www.speech.cs.cmu.edu/tools/lmtool.html). This utility lists words in units of 40 phonemes and has a database of over 125,000 words, with many words having multiple possible pronunciations. However, because our focus is on speaker dependent recognition, we use only the pronunciations that are specific to the user. Dictionary files are created for each partition, and listing all words in its vocabulary database as sequences of constituent phonemes. The unit list files list all constituent phonetic units from the dictionary files, without repetition and alongside filler phonemes that denote silence, coughing etc. Once again, unit list files are created for each partition. Next, the language model file is used to depict probabilities of sequential occurrence of words, in conformance with grammatical rules of the target language or the patterns prevalent in user speech. As we currently focus on isolated word recognition, the language model for each partition simply lists out all words in the vocabulary database, and is obtained from the language model generator utility at www.speech.cs.cmu.edu/tools/lmtool.html. Finally, sphinx requires the Audio Streams for all words in the vocabulary database and corresponding textual transcriptions for the training of models and subsequently for examining their accuracies. These are provided by the Train and Test Audio File ID's and Transcription files respectively. Previous work [37] has shown that with partitioned vocabularies we can train acoustic models with lesser training times, and higher accuracies, robustness to vocabulary changes. As the experiments we setup entail training acoustic models numbering in twenties and thirties, and we setup multiple experiments, iterative training to achieve perfect accuracies would have contributed to large experiment times, despite the simplification due to contextual partitioning. Additionally, the aim in this research is to measure recognition time performance the DALI, and benefits of CP in improving accuracies have already been established, it is thus safe to assume the availability of accurate recognition models. In line with this assumption, we use the same speech samples for both testing and training our acoustic models. This ensures perfect accuracies in recognition, without spending proportional times in training. We record two audio files per word in the dictionary using Cyber Acoustics AC-204 headset on a Lenovo T430s laptop. Each recording was approximately 5 seconds in length with some initial and trailing silence, and was encoded in 32 bit PCM quantization while sampling at 16000 Hz. The train and test file ID's are then prepared to list the paths to the recorded audio files, and the corresponding transcription files list out words corresponding to each audio stream. We do not alter the other training parameters from their default values in the configuration files. The version information of tools used is, SphinxTrain version 1.0.7. PocketSphinx version 0.7. All recognition and training procedures were performed on formed on a 2.67 GHz Core 2 Duo Desktop with 2 GB of DDR2 memory running Ubuntu Natty (11.04) 64-bit. This completes the setup of speech recognition for the DALI. Next we discuss the DALI simulator.

7.2 DALI Simulator

Existing simulators can be considered to fall into established architectural classifications of SISD, SIMD or MIMD and, as is the case for GPUs, their combinations. In real time, the DALI belongs to the SISD classification, but the dream phase operates in MISD parallelism, which is an architectural classification that has been largely ignored in previous research. As a result simulation solutions for the DALI were not readily available. Additionally, flexibility and scalability of simulation are key concerns for the DALI, since we are investigating high-level, proof of concept concerns. Also, the DALI is in an early design cycle stage, thus its architecture underwent frequent changes, necessitating small setup and evaluation times for an efficient exploration of the design space. For traditional architectures, a higher level of architectural abstraction can mean lower accuracies, because performance, even in simulators, strongly depends on contention overheads between parallel entities. And these overheads can be faithfully reproduced only at lower levels of architectural abstraction. Significantly, negligible contention exists between parallel entities, i.e., partitions in the DALI. This is because partitions operate in ideal MISD parallelism in the dream phase and only one is selected per input in real time while others remain inactive. Thus, even simulation at a high level of abstraction will have minimal impact on the DALI's simulation accuracy. We developed a software platform to simulate the DALI's functionality and calibrated it with high level performance annotations consistent with prior research efforts. The software platform for functional simulation is discussed first, followed by the methodology for performance annotations.

7.2.1 Software Platform for Functional Simulation

The DALI's architecture consists of 4 key components: Partition-Processing Element (PE) pairs, the Context Activation Processor (CAP), the Context Sensitive Network (CSN) and the Periodic Memory. Apart from these components, we must also simulate for the input stream to represent the user. To discuss the simulator setup for these components we use a subset of our example spatio-temporal hierarchy, as presented previously in Figure 19. This example subset is shown in Figure 20.

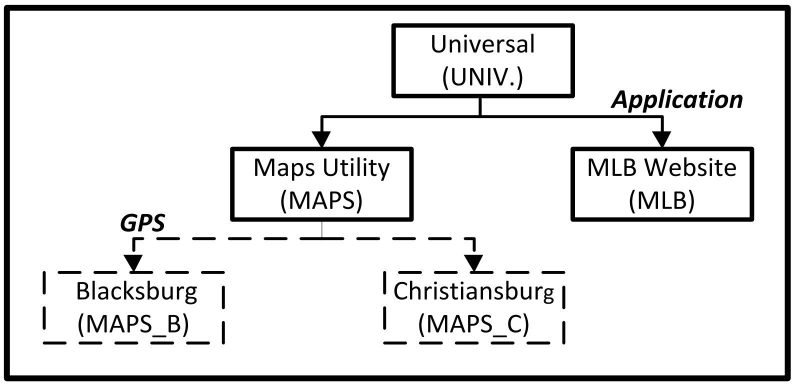


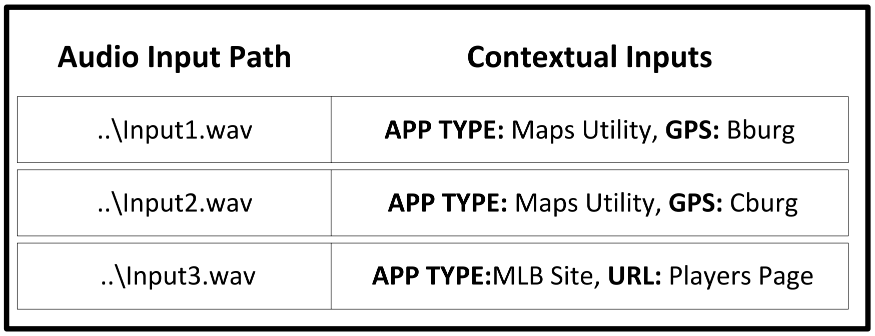
Figure 20: Example Spatio-Temporal hierarchy for Simulator Discussion

In our example spatio-temporal hierarchy, 3 partitions already exist i.e. Univ., MAPS and MLB, while two i.e. MAPS_B and MAPS_C are yet to be originated. Here, we discuss the real time and dream phase simulation for the DALI's components using this example hierarchy.

User Input Stream

The user input stream in the DALI consists of Application Inputs such as the Audio Stream for speech recognition and the accompanying contextual inputs. We simulate this with the use of an

Input (I/P) File. The I/P File lists the audio input and the contextual inputs for each real time input, in chronological order of real time arrival. Its format is visualized in Figure 21.



Audio Input Path	Contextual Inputs
..\Input1.wav	**APP TYPE:** Maps Utility, **GPS:** Bburg
..\Input2.wav	**APP TYPE:** Maps Utility, **GPS:** Cburg
..\Input3.wav	**APP TYPE:**MLB Site, **URL:** Players Page

Figure 21: Input (I/P) File and Periodic Memory (PM) File format

We consider an I/P File consisting of 3 real time inputs. The audio input is specified as the system path to the recorded file (*.wav). The accompanying contextual inputs are listed as their name (shown in bold letters) followed by their value. For inputs shown in the figure, Input 1 is received while using the maps utility as the application, with the GPS location of the user in Blacksburg. The second input is once again received when using the maps utility but the GPS location is Christiansburg. The third input is received when browsing the MLB website and the specific URL is the player's page.

Periodic Memory

The function of the periodic memory is to store real time inputs (audio input and the contextual inputs) and replay them in the dream phase. We simulated this using a Periodic Memory (PM) File. Figure 22 depicts its real time and dream phase simulation.

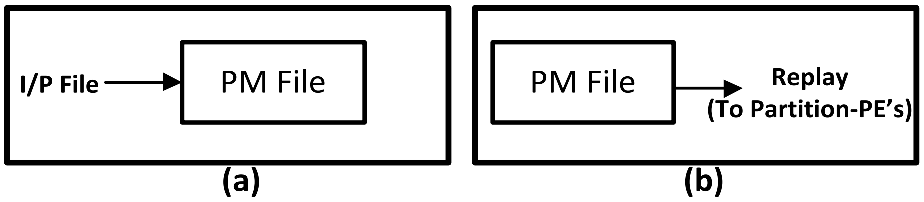


Figure 22: Periodic Memory (PM) File (a) Real Time, (b) Dream Phase

In real time, to simulate the storage of inputs, the contents of the I/P file are copied into the PM File. The PM File is thus maintained in the same format as the I/P file shown in Figure 21. To simulate replay in the dream phase, a copy of the PM file is provided to all partition-PE pairs. While ideally the contents of PM file should be an exact replica of the Input file, because all real time inputs are stored as such. For the sake of modularity of the simulator however, we decided to keep the contents of the I/P File and the PM File distinct. This gives us the ability to simulate the dream phase and real time operation for a different set of inputs, even if the dream phase and real time operation lie consecutive. This is essential as we might simulate real time operation for a smaller set of inputs than dreaming. This is because the inputs in real time are received, recognized sequentially and thus add significantly to experimentation times.

Partition-PE pairs

Partition-PE pairs in the DALI execute the speech recognition algorithm in the real time and the MCP algorithm in the dream phase. A partition-PE pair in our simulator is represented by a partition pointer. The pointer is made to reference the speech recognition algorithm for real time simulation and the MCP algorithm for the dream phase. Figure 23 depicts the real time and dream phase simulation of partition-PE pairs for our example contextual hierarchy.

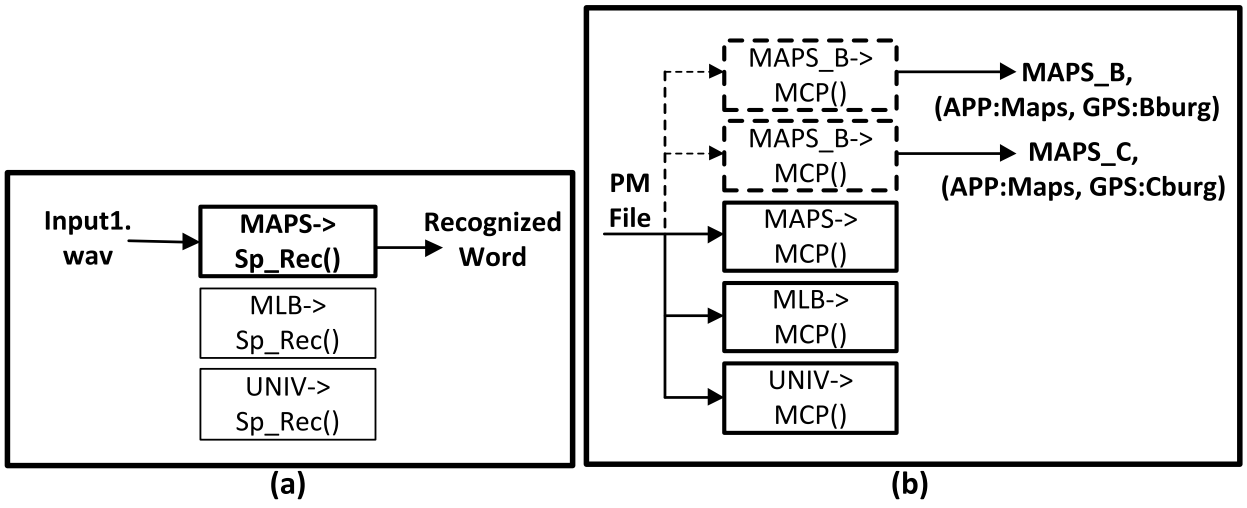


Figure 23: Partition-PE pair Simulation (a) Real Time (b) Dream Phase

We first discuss the real time simulation. The DALI initially consists of three partition-PE pairs corresponding to the existing partitions of UNIV, MAPS and MLB, and the pointers for each partition reference the corresponding speech recognition algorithm (Sp_Rec()). The CAP-CSN selects a partition-PE pair for each input listed in the Input File, and passes the audio path as an argument to the pointer of the selected partitions. For the first input in the example I/P file, MAPS partition will be selected. The MAPS speech recognizer receives the path to Input1.wav as an argument, and in turn it outputs the recognized word. A similar procedure is followed for every other input in the input file.

Figure 23(b) depicts the dream phase simulation for partition-PE pairs. Initially only 3 partition-PE pairs exist corresponding to MAPS, MLB and UNIV, as the other two partitions (MAPS_B, MAPS_C) are yet to be originated. The pointer for each existing partition now references the MCP algorithm (MCP()). To simulate the replay of real time inputs, the MCP algorithm for each partition is passed the PM file as an argument. Each partition then executes the MCP algorithm over the real time inputs recorded in the PM file, and integrates its vocabulary or originates new partitions. In our example, the MAPS partition originates two new partitions – MAPS_B and MAPS_C. Their origination is simulated by dividing the parent partitions (i.e. MAPS) vocabulary database between the two children, and generating partition pointer for each. At this point, even the newly originated partitions participate in the dream phase and receive the PMF. Once all partitions finish executing the MCP algorithm, the newly originated partitions i.e. MAPS_B, MAPS_C output a feedback structure for the re-programming of the CAP and CSN. The feedback structure consists of the partitions pointer i.e. MAPS_B, MAPS_C respectively and the set of contextual inputs used in their origination i.e. App Type: Maps utility and GPS: Blacksburg for MAPS_B.

The MCP algorithm itself had to be adapted for speech recognition. The first challenge is the choice of an appropriate Context Performance Metric (CPM). As previously discussed, the CPM defines the required level of real time performance and is used in deciding when new partitions must be originated. One option is using the required speech recognition time as the CPM. This however would result in higher simulation times, because a comparison with the CPM would require a sample recognition task to be run for every partition. A better alternative is using a vocabulary size calibrated to the required recognition time as the CPM. We refer to this as the Threshold Vocabulary Size (TVS). The reasoning is that if a partition violates the TVS it will also violate the recognition time it is calibrated to, because vocabulary size is directly proportional to recognition times. And the simulation times will be relatively smaller, because a comparison with the CPM entails a low overhead task of calculating the partitions vocabulary size. Figure 24 shows the pseudo code of the MCP algorithm developed for speech recognition.

```
For (Each Input in the PM File)
    Comparative Performance Analysis
    IF (Terminal Partition)
        Compute Vocabulary Size
        IF(Vocabulary Size > TVS) Then
            Origination Required
    Replay Analysis
    IF (Audio Input Context == Partition Context) Then
        Integrate Word in Vocabulary Database
        IF (Origination Required ) Then
            Identify Contextual Input
            Divide vocabulary between Children and generate pointers
    Re-Train speech recognizer
    Generate and Output Feedback Structure
```

Figure 24: MCP algorithm for Speech Recognition

In the comparative performance analysis, each terminal partition compares its vocabulary size with the TVS. A partition that violates the TVS is flagged as requiring origination. In our example, for MAPS_B, MAPS_C to originate, the vocabulary size of MAPS would have violated the specified TVS. The replay analysis begins by comparing the contextual inputs in the replayed input with the contextual inputs used in the partitions classification. If they match, the audio input is integrated into the partitions vocabulary. In our example, the MAPS partition will integrate the first and second input, the MLB partition the third and the UNIV partition-all three, because it is context agnostic. Next, the partitions that are flagged as requiring origination must execute the partition origination algorithm. As we have not proposed a partition origination algorithm, we simplify this by pre-programming each partition to expect a particular contextual input as per the example spatio-temporal hierarchy discussed in Figure 19. For example: the MAPS partition is pre-programmed to look for the GPS contextual input, and originate partitions in accordance with its values i.e. MAPS_B for Bburg and MAPS_C for Cburg. The vocabulary of the parent partition-MAPS is then divided between the children-MAPS_C and MAPS_B, and pointers are generated for the newly originated partitions. Next, the partitions speech recognition algorithm is re-trained, and finally, the feedback structure is generated and output for the newly originated partitions.

Context Activation Processor (CAP) and Context Sensitive Network (CSN)

In real time, the CAP-CSN together select a partition-PE pair for every input, and they are re-programmed in the dream phase. In our conceptualization of the DALI, they have been kept distinct, with the CAP responsible for scheduling a partition and the CSN for on chip communication. This distinction is necessary for optimized power performance because the CAP must remain active constantly as it monitors user context, however the CSN needs to be active only when an application input is actually received. For experimentation however it is sufficient to simulate them as a single architectural component. This is because we have not proposed novel layouts for the CSN, or implemented a predictive scheduling algorithm for CAP. Presently we only aim simulate their functionality and to this end combining them lowers simulation times.

To simulate the CAP-CSN we implement a simple partition scheduling algorithm that consists of two software components: an I/P file parser and a selection table. The parser separates each input in the I/P file into the audio path and the accompanying set of contextual inputs. The selection table is a data structure that maps partitions with the corresponding classifying contextual inputs. We implement this as a hash table, where the hashing function uses contextual inputs as the keys and the pointer to the mapped partition as the value. These sub components are depicted in Figure 25.

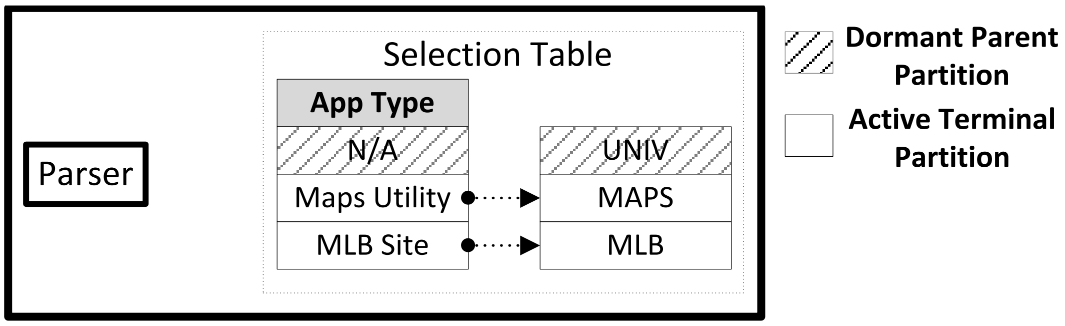


Figure 25: CAP-CSN Simulator

Initially, our example hierarchy consists of three partitions; these are UNIV, MAPS and MLB. Each of these is mapped in the selection table. The MAPS, MLB partitions are classified using Application Type, thus corresponding App Type values are mapped as keys in the selection table i.e. Maps Utility to the MAPS pointer and MLB Site to the MLB pointer. The UNIV partition is context agnostic and this is indicated by listing N/A (not applicable) as its key value. A critical point to note is that *the CAP-CSN at any point is programmed to actively select only the terminal* *partitions in real time, while the parent partitions are dormant.* For our example hierarchy, this means that in the selection table, initially, MAPS, MLB are active, whereas UNIV is kept dormant. The reason is that CAP-CSN must select only partitions that can maintain real time performance. For our simulator these are the ones with vocabulary sizes smaller the TVS, as the TVS is calibrated to required recognition time level. Thus, a parent partition is unfit for real time use, as the MCP algorithm originates children partitions only when their parent crosses the TVS. Nonetheless, entries for parent partitions must be maintained to address inaccuracies, both retirement and integration. However, as currently we consider only accurate operation; in our experiments the parent partitions will remain dormant. Figure 26 depicts the real time simulation of CAP-CSN for the example input file.

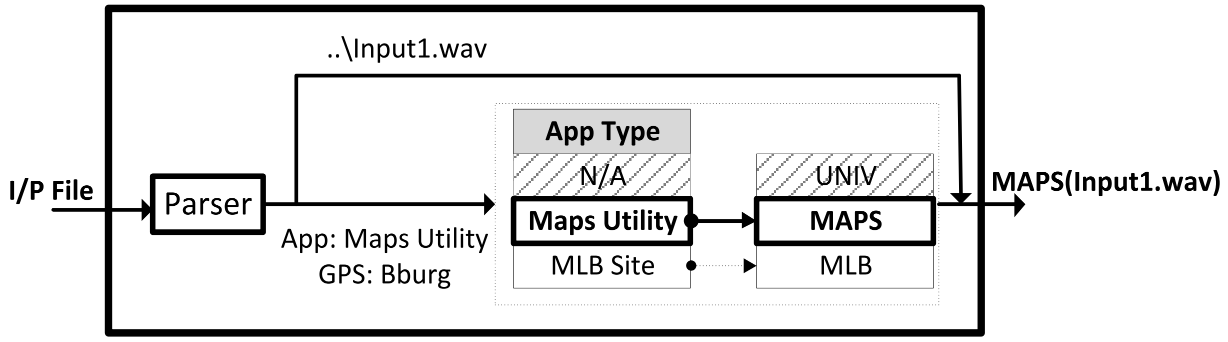


Figure 26: CAP-CSN Selection in Real Time

The parser receives the I/P file and it separates each listed input into the audio path and the corresponding set of contextual inputs. The first input is parsed into Input1.wav and the set of contextual inputs: App Type: Map Utility, GPS: Bburg. The set of contextual inputs is forwarded into the selection table to select a partition. The selection table shown in the Figure would recognize and use only the App Type as the key, while the GPS value would be ignored. Using the key as Maps Utility, it returns a pointer to the MAPS partition. The pointer is then called with the parsed audio path–Input1.wav as the argument. Figure 27 depicts the simulation of the CAP-CSN in the dream phase.

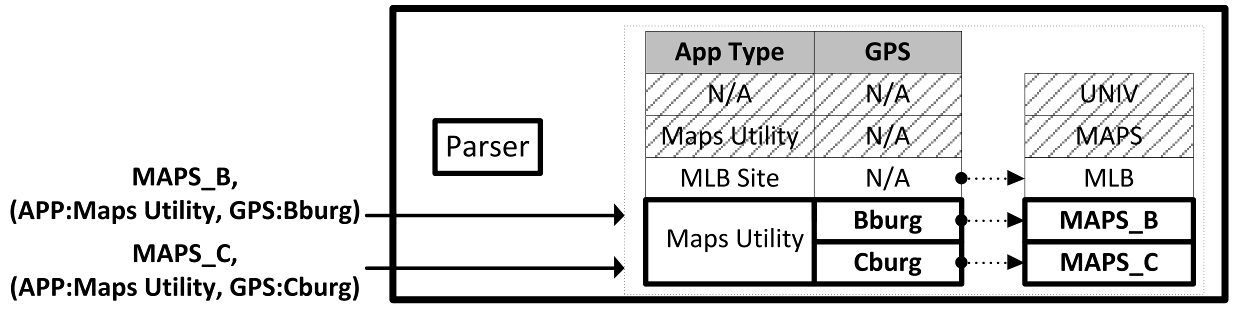


Figure 27: CAP-CSN Re-programming in the Dream Phase

In the dream phase the CAP-CSN receives feedback from the newly originated partitions- MAPS_B and MAPS_C for our example. As discussed previously, the feedback structure consists of the partition pointer and the contextual inputs used in its classification. This is used to re-program the selection table. First, the parent partitions to the newly originated partitions are made dormant for real time use – MAPS in our example. Second, entries are added for the newly originated partitions. For our example, the GPS input is added as a key and pointers are added for both MAPS_B and MAPS_C. Notably, at the end of the dream phase the CAP-CSN has been programmed to actively select from only the terminal partitions in the hierarchy (in Figure 20) i.e. MLB, MAPS_B and MAPS_C.

7.2.2 Performance Annotation

We now review the methodology for calibrating the functional simulation platform with performance annotations. Out of the two operational phases of the DALI (real time and the dream phase), we annotate performance for only real time operation. Even though the dream phase has time constraints, the time it takes for completion does not directly affect the real time recognition rates of DALI. Nevertheless, because of the ability to simplify algorithmic processing to fit a large window of time by leveraging independent, ideal MISD parallelism, the bounds of dream phase times are not a first-order concern, even against increasing algorithmic and architectural complexities. Thus, in our current research we focus on real time operation.

In the real time operation of the DALI, even as CAP, CSN play pivotal roles, their overheads on performance are minimal because of predictive scheduling and minimal on chip communication (as discussed in Chapter 6). This leaves the partition–PE pair that is selected for each audio input as the chief determinant of DALI's real time performance. To annotate its performance, initially we used the Modeling Environment for Software and Hardware (MESH), a high-level, layered software-on-hardware modeling environment [101-103]. Eventually, we transitioned to using System Time for making these annotations, as it offers lower setup, development times, easier replication in future research and better accuracies in comparison to MESH. We first summarize the background and calibration of MESH, and subsequently for System Time annotations.

Modeling Environment for Software and Hardware (MESH)

The purpose of MESH is to enable the architects to make high-level, multi-core architectural design decisions, by allowing for fast and accurate evaluation of the impact of trade-offs in Single Chip Heterogeneous Multiprocessor (SCHM) architectures [101-103]. It has been shown to achieve accuracies up to 80% better than other high level tools, such as analytical models, while offering simulation speeds almost 100x those of Instruction level simulators [101]. It adopts a layered approach to model SCHM's, shown in Figure 28.

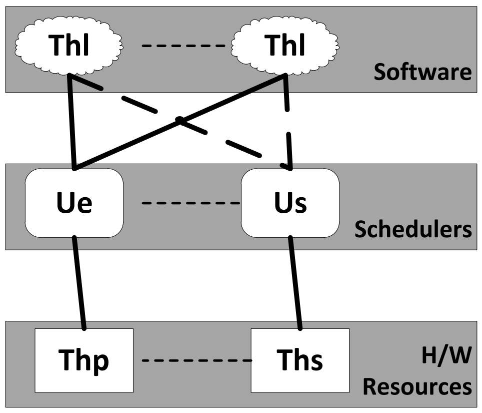


Figure 28: MESH Modeling Environment

The layered approach consists of a layer of logical threads (Thl) that models software, and executes over a layer of schedulers (Ue) that interface the software threads onto the multiple heterogeneous hardware resources (Thp) in the physical thread layer. The computational complexities of logical threads are expressed by annotating their code with consume calls, and the software can be partitioned into annotation regions at desired levels of granularity. The variable size of annotation regions allows for modeling performance starting from instruction level and higher. It is recommended that the size of annotation regions and the consume call values be determined by profiling the application algorithm or using previously benchmarked performance [101]. Different physical threads i.e. the hardware resources can be defined using distinct computational powers (computation per unit time). The scheduling layer then resolves the ordering of software onto the hardware based on the software annotations and available physical computational power. Further even the scheduling algorithms can be annotated with overhead values. This layered model can be used for modeling performance at different levels of granularity in accordance with research aims, while posing minimal setup overheads compared to instruction or cycle level modeling.

On the DALI's architectural framework the CAP is responsible for scheduling, however as discussed in the introduction to this section its overheads can be safely abstracted without affecting accuracies. This means that to integrate MESH into the DALI simulator, we must annotate the software complexity of speech recognition algorithm corresponding to each partition and the computational powers of the corresponding PE. As we presently consider a homogeneous set of PE's, all the PE's were defined with the same computational power. MESH can make software annotations at many levels of abstraction and for our goals high level annotations are sufficient. Specifically the annotation used was 1 MESH cycle for every word in the speech recognizer's vocabulary database. These annotations are based on previous research that has shown a direct dependence of recognition times on the size of the vocabulary database [104]. While differential phoneme contents in words will result to some variations, these will be minimal when considering the average performance across all words in the vocabulary. This formed the setup for early experiments; however we eventually shifted to using system time for annotating the simulator. The reasons underlying this transition and the corresponding setup are discussed next.

System Time

An alternative option for performance annotations is using the System Time taken by the selected partition to recognize the input audio. This assumes the desktop processor to be the PE allocated to the selected partition. Its benefits over MESH are better scalability and lower setup times, as minimal software development is needed, even in comparison to a high level tool such as MESH. This also makes it easier to replicate the DALI simulator in future research efforts. Additionally, it offers greater accuracy compared to MESH. This is because in MESH we annotate based on estimated speech recognition complexities; whereas system time actually executes the speech recognition algorithm and measures its performance. It will thus capture algorithmic-architectural interactions more faithfully if the processor type in the system being used is consistent with the PE types anticipated in an instance of the DALI. Significantly, in real time, the DALI uses a single PE per audio input, as other PE's remain inactive. Thus despite having multiple PE's, the real time operation is faithfully represented by simulating for just one selected PE per input, justifying the use of system time. Another argument against system time is that architectures for mobile devices are comparatively more resource constrained than desktop units. However, we compare recognition time values at different vocabulary sizes, between themselves, and no comparisons are drawn with previously benchmarked speech recognition results over other architectures.

To annotate, we use clock_gettime () (from the system.h) library to measure start, end execution time of a partitions speech recognizer, and the delta between the two measurements to get the system time consumed by the partition. This gives us a granularity in nanoseconds, which suffices for our setup, as recognition time values with our vocabulary database were measured within a hundred milliseconds. Prior to using system time, we had to ensure that the layer of operating system software between the functional simulation platform and the desktop processor does not vary system time measurements. For this, we examine the consistency of system time measurements, and ensure that it extends to increasing vocabulary sizes, by executing the following experiment. Out of the vocabulary database discussed in the previous section, we select words at random to create 6 smaller vocabulary databases starting at 20 words up to 120 words, incrementing 20 words for each database. Acoustic models were then trained for all 6 vocabulary databases, and speech recognizers trained for them. These are then used to recognize all words in the corresponding vocabulary database, and 50 system time measurements are collected for every word. We then calculate the Standard Error (SE) value for the collected samples. The Standard Error (SE) of the mean indicates the uncertainty surrounding the measurement of the mean value [105, 106]. In other words, if the standard error values are large the average cannot be used as a true representative of entire population of the sample. Table 4 lists the Mean values and the calculated Standard Error (SE) values for 4 words. We do not include the data for the entire vocabulary, as the rest of the words show similar results and their inclusion would have been an unnecessary addition to the length of the text.

Vocabulary Size	Word 1		Word 2		Word 3		Word 4	
	Mean	S.E.	Mean	S.E.	Mean	S.E.	Mean	S.E.
20	152.13	1.53	152.46	1.48	145.12	1.37	157.81	1.38
40	169.48	1.12	158.56	1.39	159.79	1.23	170.81	1.41
60	183.47	1.46	170.34	1.25	172.64	1.18	184.29	1.27
80	209.07	1.61	184.23	1.44	189.05	1.39	215.58	1.64
100	226.4	1.73	202.79	1.57	202.48	1.62	235.84	2.07
120	250.37	1.8	230.69	1.74	243.13	1.81	261.95	1.39

Table 4: Standard Error (SE) of the Mean for System Time Measurements

The standard error values for each word are small, generally lying within 1 % of the mean values. These values are extremely small especially in comparison to the increment between vocabulary sizes. Further, the standard error values do not show a monotonic increasing or decreasing pattern between vocabulary sizes, instead remain approximately stable. Our vocabulary database is sufficiently varied, and covers the standard range of phonemes (speech units) as used in other vocabulary databases, we are thus confident that this stability is not specific to our platform, and can be replicated by other accurately trained speech recognizers. Overall this experiment shows that the average system time value is consistent and there is negligible interference by the operating system in its measurement.

Next, as MESH formed the baseline for performance modeling, we examined the conformity of system time measurements with MESH. This is done to ensure that the value measured using system time is compliant with our annotations made in MESH, and in this way examine their accuracies. To do this, we setup and execute the following experiment. We measure MESH cycles and System Time for speech recognition as vocabulary is increased from 20 to 220 words in steps of 20 words. As in the earlier experiment, vocabulary databases are created for every vocabulary size, and acoustic models are trained on them. The corresponding decoders are then annotated with clock_gettime () calls to measure system time, while MESH cycles are measured using the setup discussed in the preceding sub section. The DALI's performance at each vocabulary size is measured as the average of system times measured for all words in the database, and 50 system time samples are collected for every word. Figure 29 plots the obtained performance of both MESH and system time against rising vocabularies.

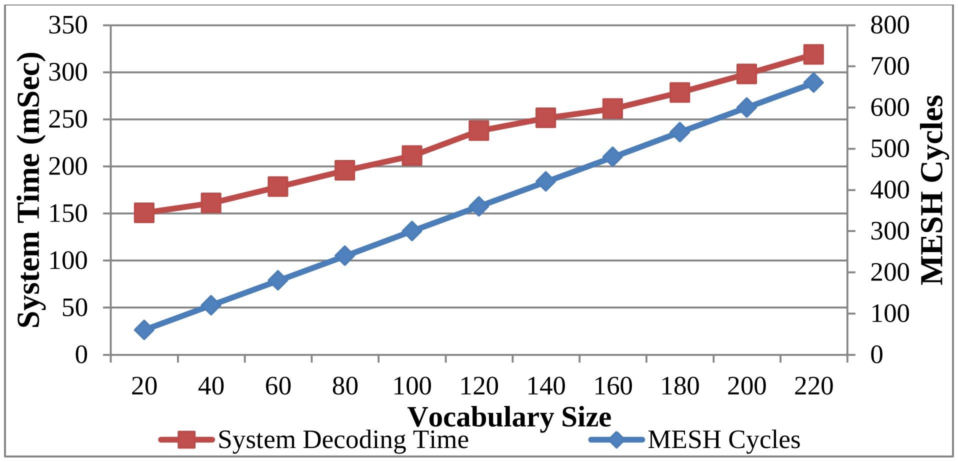


Figure 29: Comparing Performance Annotations between MESH and System Time

Both MESH and System Time performance increase along a linear curve against vocabulary sizes. While the slopes of the curves differ, the noticeable point is that as per our annotations for MESH, there is a linear relationship between the vocabulary size and the measured system time values. This proves that the measurements made using system time are consistent the baseline MESH annotations, and thus the use of system time for modeling the DALI's recognition time performance is a viable alternative to MESH.

7.3 Summary

In this chapter, we have reviewed our experimental platform setup to examine the DALI's performance for example application to speech recognition. Our goal is to examine if the DALI can maintain constant real time recognition rates against increasing vocabulary sizes and partition-PE pair numbers. We developed a speaker dependent vocabulary database personalized to a hypothetical user personality, and defined a set of partitions in the database, in accordance with plausible lifestyle of the hypothetical user. We used CMU's Sphinx Recognition suite to setup speech recognition over the vocabulary database. Finally, in the absence of readily available simulators, we developed a high level architectural simulator for the DALI. In the next chapter, we discuss the experiments setup over this experimental platform.

8 Experiments

We set up experiments and executed them using the DALI experimental platform developed for this research. The goal is to evaluate if the DALI can maintain constant recognition time performance against increasing vocabulary size and PE numbers. We identify the following experimental variables:

- Required recognition time
- Number of Processing Element's (PE's)

Recognition times for a speech interface must match the user's rate of speech, while recognition any faster is un-necessary. The user's rate of speech thus defines the required level at which the DALI must maintain constant recognition time. However, speech rate, even for the same user, varies. For example: the user might speak much faster while dictating email at work compared to when browsing the internet at leisure. To account for speech rate variation we use the required recognition time as the first experimental variable. We then examine if the DALI can maintain the required recognition time as vocabulary sizes and PE numbers grow, which is consistent with our definition of scalability, previously discussed. The vocabulary size however depends upon the user, and a limit cannot be defined at design time. However, the number of PE's on a SoC is fixed at design time, and the DALI must accommodate the user's vocabulary within this number. Also, the number of PE's increases with every generation of SoC architectures [57]. Thus, to examine the DALI's scalability with future SoC architectures, we use the number of PE's as the second experimental variable. Three experiment sets are setup and simulated with the use of these variables. Here we first discuss the setup of the experiments sets, and subsequently their simulation.

8.1 Setup

We first define the experimental variables for each experiment set. Subsequently, we discuss the common simulation procedure followed in each experiment set. Apart from the DALI's simulation, we also simulate for a speech recognizer with un-partitioned vocabulary database over the same vocabulary sizes as the DALI. This is important because previous work in CP has compared only the training times of contextually partitioned and un-partitioned databases, and the comparison of recognition time performance remains.

8.1.1 Defining Experiment Variables

To set the required recognition time value, we use the human rate of speech as a baseline. Human rate of speech lies between 160-200 words per minute (wpm) [107], which covers speakers with average speed (for example, books on tape or orators and on the higher side, speed speakers such as auctioneers). In the simulation setup, recognition times are measured over a desktop processor, whereas the DALI targets mobile devices which are comparatively more resource constrained. To compensate for the difference in computational resources, we scale up the range of speech rates in our experiments, to lie between 350 and 400 wpm. While higher than the actual rates, it is still in the same order of magnitude, which seems a reasonable assumption to make. For each set then we choose a speech rate from this range, and the required recognition time is calculated as the inverse of the speech rate value.

In current mobile devices, architectures with 4 or more PE's are common, and state of the art designs integrate up to 8 PE's. Experts expect this number to continue to increase in future designs, with numbers doubling within the next couple of generations [57]. As our intention is to show that DALI can scale well beyond levels of traditional parallel architectures, we experiment with PE number in excess of 8. Performance benefits of most traditional parallel architectures degrade beyond 6-8 PE's. Thus 8 forms the lower limit for PE numbers in our experiments. Because each PE is allocated to a partition, the upper limit of the PE number is defined by the maximum number of partitions possible in our vocabulary database. In Chapter 7, we defined this number as 16, thus the range of PE numbers for our experiments is between 8 and 16.

We setup three experiment sets to cover the defined range of speech rates and PE numbers. The experimental variables chosen for each are as follows:

Experiment Set 1: To form a baseline, in the first experiment set we consider an average use case scenario. For this, we consider a user with average rate of speech and PE numbers corresponding to a device from the current generation. The upper and lower limits of the speech rate range would correspond to slow and fast speakers respectively, and it is reasonable to consider that an average user will have a speech rate in the middle of this range i.e. around 375 wpm. This corresponds to a required recognition time value of 161 msec. State of art mobile devices today, integrate up to 8 PE's (Table 2) and with multi-core scaling these will soon be commonplace and it is reasonable to assume that an average user will own a device with 8 PE's. Thus, the number of PE's for experiment set 1 is defined as 8.

Experiment Set 2: Moving on from the average use case scenario, in this experiment we cover the lower end of the range of speech rates, a plausible scenario for a user who speaks slow either inherently or because the application/device requires infrequent enunciation. For this we use the lower end of the speech rate range-350 wpm, which translates to a required recognition time value of 172 msec. In order to examine DALI's scalability against a larger number of PE's, we increase their number compared to the previous experiment. PE numbers generally rise in multiples of 2 within generations of devices. We thus specify the PE number at 12 for experiment set 2, an increase of 4 over the previous set.

Experiment Set 3: Finally, having covered slower and average rates of speech in the previous two experiments, upper limit of the range of speech rate is left uncovered. To include this, in the final experiment set we define the speech rate at the upper limit i.e. 400 wpm, which translates to a required recognition time of 150 msec. TVS upon calibration to required recognition time-150 msec, settles at 22 words. We move this slightly higher to 25 words, and proportionally re-adjust the required recognition time to 155 msec. This is because, between previous two experiments, the TVS values differ by 10 words (35 for the first and 45 for the second). For a better comparison with previous sets, we wanted to maintain the same TVS difference between experiment set 1 and experiment set 3 as well. For PE numbers, we continue to scale over the previous experiment set, and. add another 4 to reach the upper limit of our experiment range at 16 PE's.

The expectation in all the three experiment sets is that the DALI will maintain recognition time constant at the respective required level, as vocabulary size increases and it leverages all available PE's. Next, we discuss the simulation procedure in the three experiments.

8.1.2 Simulation Procedure

We first discuss the simulation for the DALI and then the un-partitioned recognizer.

DALI

The steps followed for the DALI's simulation are:

1. ***TVS Calibration:*** The MCP algorithm (Figure 23) uses the TVS to decide when partitions must be originated. The first step is calibrating its value to the required recognition time for the experiment set. The procedure followed is:
    - A threshold vocabulary database is developed and a speech recognizer is trained over it. The database consists of random words selected from the vocabulary database.
    - Its recognition time is measured using all audio inputs in its database, and depending on whether the recognition times are higher or lower than the required recognition time, its vocabulary is incremented or decremented.
    - This process is repeated until its recognition time matches the required value, and the corresponding size of the threshold vocabulary database is then used as the TVS.
2. ***Vocabulary Increment:*** We start with an empty vocabulary database, and increment it over multiple experiment steps until further increase in vocabulary would result in a partition count larger than the available number of PE's for the experiment set. The procedure is:
    - For each step we choose a new set of words- the New Vocabulary Set (NVS) from the experiment vocabulary database. For all experiments we set the size of NVS at a common value of 20 words, to allow for an easier comparison of results between the experiments
    - The NVS contents are chosen at random from the vocabulary database, while taking into account two plausible user characteristics: First, vocabulary growth is not monotonic along a partition. Instead, it is spread out across numerous contextual situations. Second, it is fair to assume that a user will encounter some contextual scenarios with greater probability to others, resulting in larger databases for them. To encompass these characteristics, NVS contents are chosen from all partitions in the experiment vocabulary database; however their number is kept in proportion to the relative vocabulary sizes of the partitions.
3. ***Dream Phase Simulation:*** Next, for each step we simulate the dream phase. A more detailed discussion is presented in Chapter 6, here we summarize only the key steps:
    - The periodic memory (PM) file is created to include audio stream paths and associated contextual inputs for all words in the NVS.
    - Each partition-PE pair then executes MCPA over the PMF, to integrate words into its database, originate new partitions and re-program CAP-CSN accordingly.

4. ***Real Time Simulation:*** With the new set of words integrated into DALI's vocabulary and the architecture appropriately re-organized, we commence with real time simulation. For a more detailed description refer to Chapter 6, only the key steps are summarized here:
   - We construct the Input File to include contextual cues and audio stream paths for each word that is input. The words to be input are chosen from the existing vocabulary database of DALI. In order to preserve simulation scalability, we choose a subset of the available vocabulary database-the Input Vocabulary Set (IVS). The size of IVS is fixed at 20 words for all experiments, so as to enable comparisons between experiment sets.
   - In choosing IVS contents we must ensure that the choice of words represents the vocabulary distribution correctly, and is not biased towards a particular partition. For this, similar to the NVS, the number of words chosen from a partition is kept proportional to the ratio of its vocabulary size compared to others.
   - The Input File is then parsed by CAP-CSN to select PE-partition pair, which in turn recognizes the input audio.

Un-Partitioned Recognizer

The motivation for simulating a speech recognizer with un-partitioned vocabulary database is to analyze comparative benefits of contextual partitioning, by comparing its performance with DALI. The simulation for the un-partitioned recognizer is simple, as no partitioning is required, and a single PE, carrying a speech recognizer trained over the entire vocabulary database recognizes in real time. At each step, the un-partitioned recognized shares the same vocabulary database as DALI. For a valid comparison, at each step, the un-partitioned recognizer is input with the same set of audio streams as DALI i.e. the Input Vocabulary Set (IVS) for the respective step. This culminates the setup of experiments, and next we discuss the simulations for each experiment set.

8.2 Experiment Set 1

The required recognition time for experiment set 1 was defined as 161 msec and the number of PE's as 8. Figure 30 depicts the simulation.

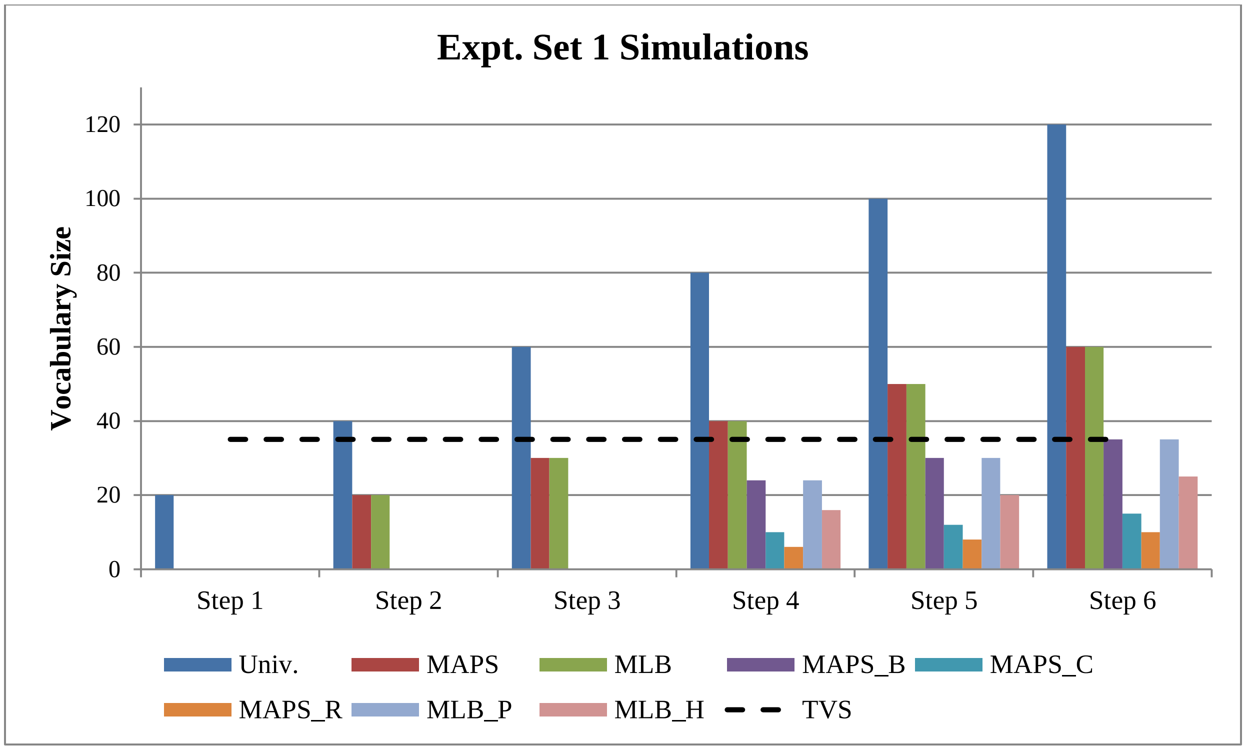


Figure 30: Experiment Set 1 Simulations

For each experiment step we plot the vocabulary size of partitions that exist after the dream phase simulation, as bar graphs. The terminal partitions can be identified as those whose names are not a subset of another partitions name at the same step, such as MAPS, MLB at Step 2 (Chapter 7 presents a deeper discussion on the identification of parent, child and terminal partitions). The TVS is plotted as a dotted line running across the experiment steps. We have two major expectations: First, the number of partitions will increase as the vocabulary size rises across the steps, indicating that the DALI's architectural complexity is coupled with algorithmic complexities. Second, despite the increase in vocabulary size, the DALI will maintain terminal partitions below the TVS line, for real time selection. This will indicate that the DALI's real time complexities remain simplified despite an overall increase, enabling constant recognition time of 161 msec, in achievement of our goals.

The TVS value was obtained as 35 words by calibrating to the required recognition time of 161 msec. The simulation runs for 6 steps over which all 8 partitions originate to utilize all available PE's. Each step adds 20 words to the DALI's vocabulary, thus the final vocabulary size achieved is 120 words. The number of bar graphs i.e. the number of partitions-PE's increase across the steps as vocabulary size rises. Interestingly, partitions originate only when an existing partition crosses the TVS line. Also, at these steps, the originated partitions themselves lie below the TVS line, while and together covering the vocabulary of the violating partition. For example: UNIV at Step 2 grows to 40 words, and MAPS, MLB originate both with vocabulary size of 20 words. The reason is that a partition larger than the TVS will recognize slower than the required recognition time of 161 msec, and its vocabulary database is thus simplified. In the terminology of the contextual hierarchy, the violating partition assumes the role of a parent, whereas the children partitions are terminal to the hierarchy. The DALI is consequently re-organized to make the parent partitions dormant and instead actively select from the new set of terminal partitions. This follows that, the partitions selected by the DALI in real time-the terminal partitions, can be identified as the ones below the TVS line at each step.

For a better understanding of origination and the consequent re-organization, Figure 31 visualizes the simulator state at each experiment step by depicting the corresponding contextual hierarchy and the CAP-CSN selection table.

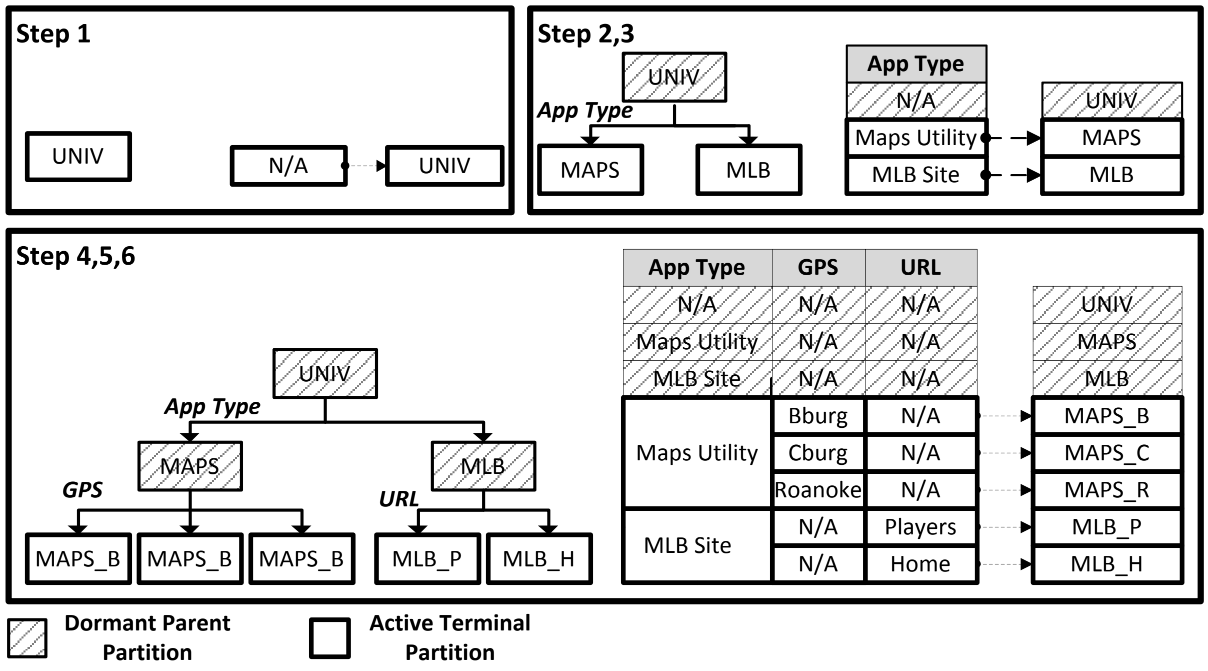


Figure 31: Contextual hierarchy and CAP-CSN Selection Table for the 6 Experiment Steps

The contextual hierarchy is drawn on the left and the corresponding CAP-CSN selection table on the right. This visualizes the simulator state for the steps shown in Figure 30. At step 1 only one partition i.e. UNIV exists, because at 20 words it is smaller than the TVS. As Univ. is context agnostic, it is not mapped to any specific contextual input (N/A) in the CAP-CSN selection table, meaning that is selected for every real time input. At Step 2, looking back at Figure 30, UNIV has crossed the TVS and as a result MAPS, MLB are originated. Now, UNIV is a parent partition, while MAPS, MLB are terminal. Accordingly, the CAP-CSN is re-programmed to make the entry for UNIV dormant and key-value pairs for MAPS and MLB are added and made active for real time selection. No new partitions originate at Step 3 because MAPS and MLB remain smaller than the TVS. Thus, the DALI maintains the same contextual hierarchy and the CAP-CSN selection table. At step 4 both MAPS and MLB cross the TVS. As a result, they originate MAPS_B, MAPS_C, MAPS_R and MLB_P, MLB_H respectively, all of which lie below the TVS line in Figure 30. This set of partitions is now terminal to the hierarchy, whereas the MAPS, MLB partitions are parents to them. The CAP-CSN selection table is accordingly re-programmed to make MAPS, MLB dormant and add, activate the new set of terminal partitions for real time selection. For the final 2 steps the contextual hierarchy and the CAP-CSN selection table remains the same, because the existing set of terminal partitions remains below the TVS. In summary, the partitions selected in real time by the DALI in real time i.e. the terminal partitions can be identified as the ones lying below the TVS line at every step.

Additionally, we expect that the terminal partitions at every step would cover the entire vocabulary database of the DALI. The reason is that children partitions always cover the entire vocabulary database of their parents, and terminal partitions lie at the bottom of the hierarchy meaning that they must encompass vocabulary of all of the parents above. To verify this, Table 5 lists the set of terminal partitions and their vocabulary sizes for every experiment step.

	Terminal Partition Vocabulary Sizes					**Sum**
Step 1	**UNIV**					**20**
	20					
Step 2	**MAPS**	**MLB**				**40**
	20	20				
Step 3	**MAPS**	**MLB**				**60**
	30	30				
Step 4	**MAPS_B**	**MAPS_C**	**MAPS_R**	**MLB_P**	**MLB_H**	**80**
	24	10	6	24	16	
Step 5	**MAPS_B**	**MAPS_C**	**MAPS_R**	**MLB_P**	**MLB_H**	**100**
	30	12	8	30	20	
Step 6	**MAPS_B**	**MAPS_C**	**MAPS_R**	**MLB_P**	**MLB_H**	**120**
	35	14	10	35	25	

Table 5: Terminal Partition Vocabulary Sizes

The first observation is in line with what we observed in the figure, in that the vocabulary size of each terminal partition at every step is less than or equal to 35 words-the TVS. More interestingly, as expected, the sum of the sizes of terminal partitions equals the overall vocabulary size. For example, at Step 1, UNIV is the only terminal partition and its vocabulary equals the overall vocabulary size of 20 words. At step 2, MAPS and MLB-the two terminal partitions individually have a size of 20 words and together they add up to 40 words-the overall vocabulary size of the DALI. Similarly, for the other four steps the terminal partitions sum up to 60, 80, 100 and 120 words-the DALI's vocabulary size for each step. This means, that the DALI's vocabulary database is always covered by the set of terminal partitions, each of which is individually smaller than 35 words. Thus, irrespective of the vocabulary size of number of partition-PE pairs, the DALI will use less than or equal to 35 words for every real time input.

The most significant observation in the figure is that the DALI's architectural complexity increases with algorithmic complexity, as new partitions, smaller than the TVS originate to replace the partitions that cross the TVS line, for real time selection. As a result, the partitions selected in the real time (terminal partitions) remain smaller than 35 words-the TVS, even as the DALI's vocabulary rises to 120 words. In this way, the DALI keeps its real time complexities simplified, despite an overall growth. We expect that this will allow the DALI to maintain recognition time constant at 161 msec, as the TVS is calibrated to this value.

8.3 Experiment Set 2

We defined the required recognition time at 172 msec and the number of PE's as 12 for the experiment set 2. Figure 32 depicts the simulations.

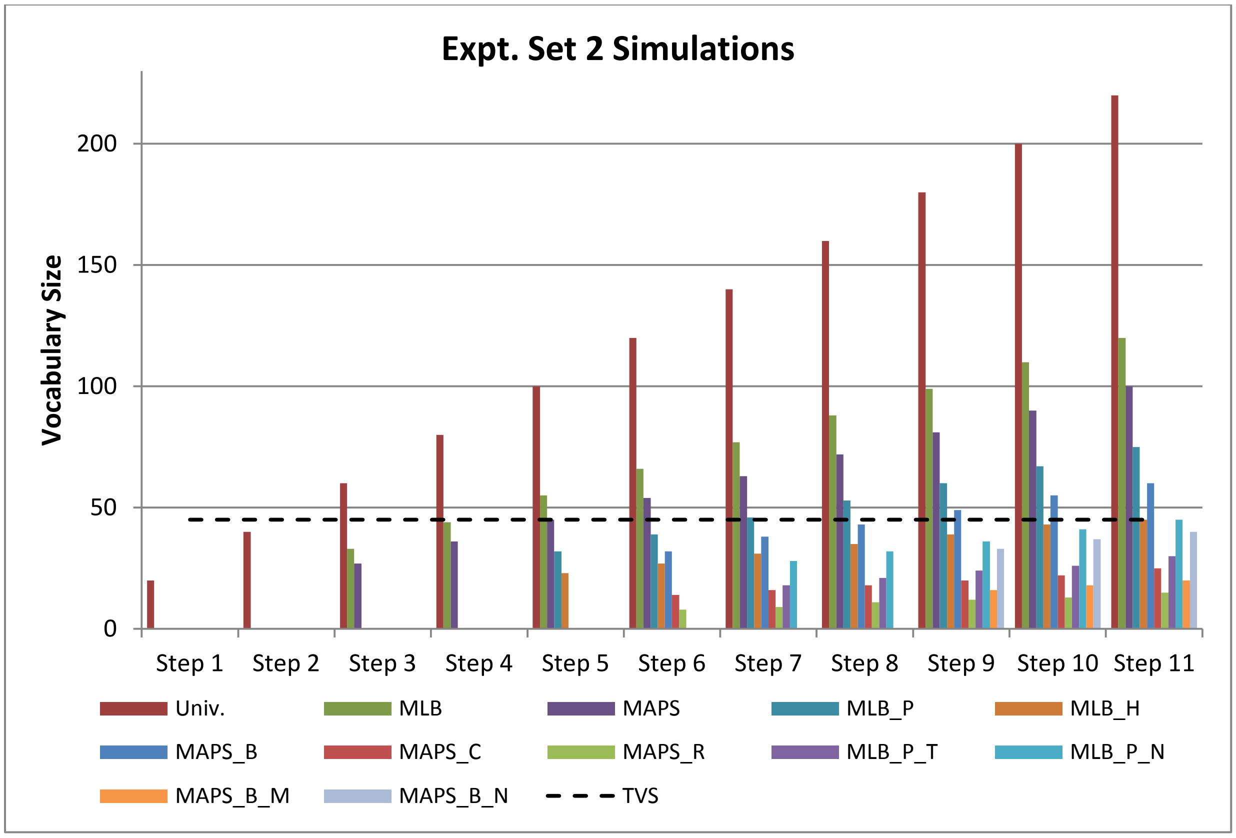


Figure 32: Experiment Set 2 Simulations

We plot the vocabulary size of partitions that exist after the dream phase as bar graphs for each experiment step. The terminal partitions can be identified as those whose names are not a subset of another partitions name at the same step. The TVS value is plotted as a dotted line running across all the steps. We have two major expectations: First, as vocabulary increases across the steps, the number of partitions will grow alongside, however slower than the previous experiment because of a comparatively slower required recognition time. This would indicate that the DALI's architectural complexity grows with algorithmic complexities, and the rate of growth is proportional to the performance requirements. Second, consistent with the previous experiment, the DALI will maintain terminal partitions below the TVS, to indicate that its real time complexities remain simplified, and enable constant recognition at 172 msec.

The TVS was calibrated to the required recognition time of 172 msec and settles at a value of 45 words. This is 10 words greater than the first experiment set which is expected because the required recognition time value is also ~ 10 msec greater (172 msec compared to 161 msec respectively). The simulation runs for 11 steps over which all 12 partitions originate to utilize all available PE’s. Each step adds 20 words to the DALI’s vocabulary, thus the DALI achieves a final vocabulary size of 220 words. We first summarize the observations that are consistent with the previous experiment. For a lengthier discussion of these observations, the previous sub-section can be referenced. The number of partitions and consequently the number of PE’s used by the DALI increases with vocabulary size. However, new partitions originate only when an existing partition crosses the TVS line. For example: UNIV at Step 3, MLB at Step 5. The originated partitions in turn are smaller than the TVS and together cover the vocabulary of their parent. For example: at Step 3, MAPS is at 33 words and MLB at 27 words, summing up to 60 words-the size of the UNIV partition. At these steps, the violating partition assumes the role of a parent and is made dormant for real time use, while the newly originated partitions are terminal to the hierarchy and are activated for real time selection. Because of the origination and the resultant re-organization, in the real time the DALI selects from the terminal partitions, which can be identified as ones lying below the TVS line at each step. Furthermore, the set of terminal partitions together cover the entire vocabulary database of the DALI at every step. For example: at Step2 UNIV is the only terminal partition and it equals the overall vocabulary size of 40 words, similarly, at Step 3, MLB and MAPS add up to 60 words-the vocabulary database size. This means that the DALI’s overall vocabulary database can be completely represented by a set of partitions with vocabulary size below or equal to 45 words-the TVS.

A notable difference from experiment set 1 is that the partitions seem to originate comparatively slower. For example: at Step 2, experiment set 1 already has 3 partitions compared to 1 in experiment set 2. Similarly, experiment set 1 has 8 partitions by step 4 compared to 3 for experiment set 2. This, even as the rate of vocabulary growth is the same i.e. 20 words per step. The underlying reason is that required recognition time is 10 msec slower in experiment set 2 compared to the 1st, and proportionally the TVS value is 10 words greater (at 45 words compared to 35 words in the 1st). This allows the partitions to grow 10 words larger than in the first experiment set before new partitions originate and resulting in a slower rate of origination and PE usage. In addition, in the experiment set 2 the DALI leverages 4 extra PE’s over and above the 1st experiment set. This, alongside the slower rate of PE usage allows the DALI to scale to 100 words more compared to the first experiment set.

The most significant observation is consistent with the previous set in that the DALI's architectural complexity rises with algorithmic complexity, as partitions that cross the TVS are replaced by newly originated partitions for real time selection. As a result, the partitions selected in real time (terminal partitions) always lie below 45 words-the TVS value, and in this way keep the DALI's real time complexities simplified, despite an overall increase. Because the TVS value is calibrated to required recognition time, we expect that the DALI will maintain performance constant at 172 msec. The most notable difference from the previous experiment is that the number of partition-PE's increases comparatively faster with vocabulary size. We explained that the reason is the comparatively higher value of required recognition time of 172 msec. This, alongside the availability of 12 PE's (i.e. 4 more than the previous experiment), allows the DALI to scale to a larger vocabulary size of 220 words.

8.4 Experiment Set 3

For experiment set 3 we defined the required recognition time as 150 msec and the available PE numbers as 16. Figure 33 depicts the simulations.

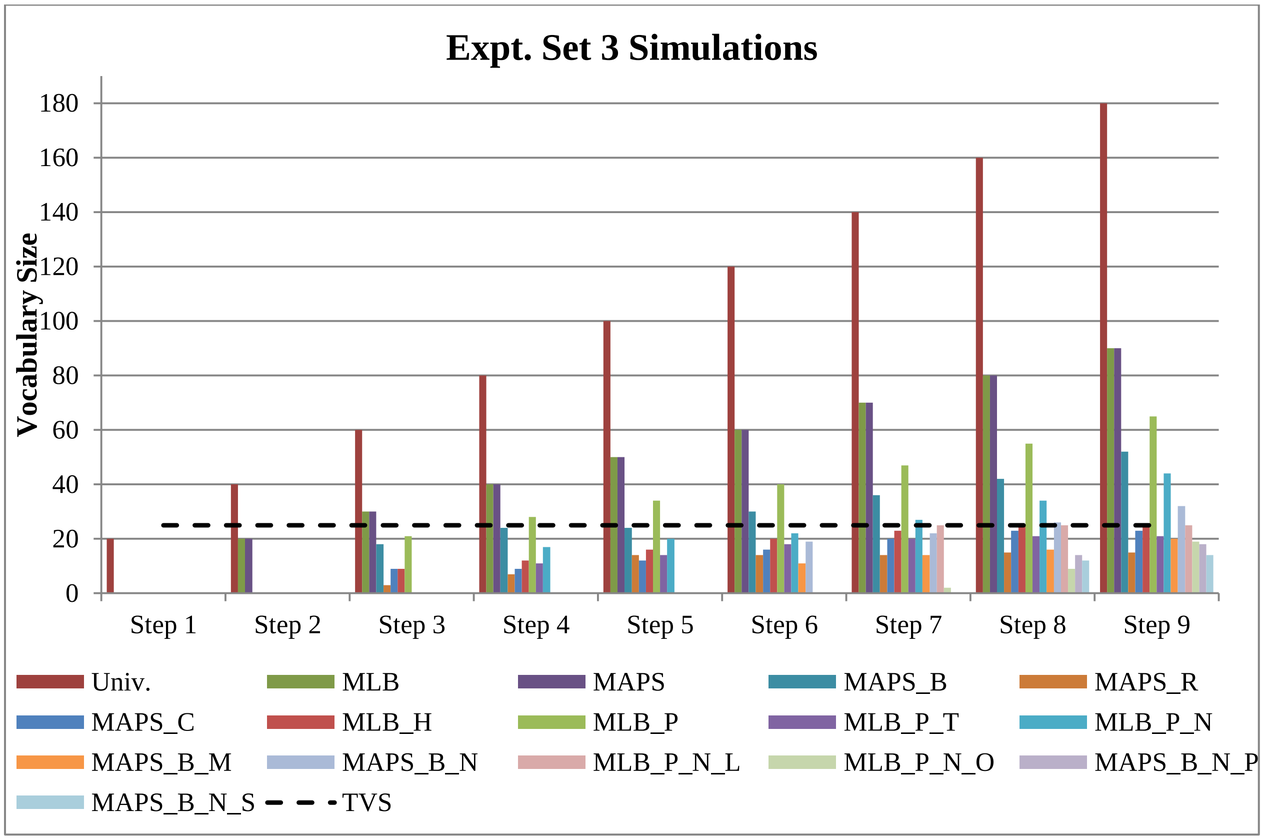


Figure 33: Experiment Set 3 Simulations

We plot the vocabulary size of partitions that exist after the dream phase as bar graphs for each experiment step. The terminal partitions can be identified as those whose names are not a subset of another partitions name at the same step. The TVS value is plotted as a dotted line running across all Steps. The two major expectations are: First, as the vocabulary grows across the experiment steps the number of partitions will increase, and at a faster rate than the previous experiments, because of a comparatively faster recognition time requirement. This will indicate that that the DALI’s architectural complexities grow with algorithmic complexities, and will confirm that the rate of increase is proportional to the performance requirements. Second, consistent with the previous experiments, the DALI will maintain terminal partitions below the TVS, to indicate that real time complexities remain simplified and enable constant recognition time of 155 msec.

The TVS was calibrated to the required recognition time of 150 msec and settled at a value of 22 words. This is smaller than both previous experiment sets (35 and 45 words respectively), which is expected because the required recognition time level is also the lowest. However, we increase this slightly to 25 words, in order to maintain symmetry with the 10 word difference between the TVS for previous experiments. The required recognition time is then re-calibrated to 25 words and settles at 155 msec. The simulations runs for 9 steps over which all 16 PE’s are used and the DALI scales to 180 words. We first summarize observations consistent with the previous sets, for a deeper discussion of these the reader can reference sub-section 8.2. The number of partitions for the DALI and consequently the number of PE’s used rises with vocabulary size. Children partitions originate only when their parent partition crosses the TVS line. For example: at Step 2 UNIV partition, at Step 3 MAPS, MLB partitions and at Step 4 MLB_P. The originated partitions themselves lie below the TVS line and together covering the vocabulary of the violating partition. For example: at Step 2, MAPS, MLB are both at 20 words and together cover UNIV partitions vocabulary. Alongside origination, the DALI is re-programmed to make the parent partition dormant and add, activate the originated terminal partitions for real time selection. Effectively, as a result of the origination and the resultant re-organization the set of partitions selected in the real time-the terminal partitions, are the ones lying below the TVS line for every step. Furthermore, because each set of children partitions together sums up to the vocabulary of their parent, the terminal partitions together cover the entire vocabulary database of the DALI.

A notable difference from both previous experiments is that the rate of increase in partition-PE numbers is faster in this experiment. This even as the rate of vocabulary increase is the same for all three experiments at 20 words a step. For example: by step 3 already 8 PE's have been used, compared to 3 in both experiment 1 and 2. At step 4 this number rises to 10, whereas experiment set 1 and 2 are both still at 3. As discussed in the previous experiment, the rate of partition origination in the DALI is directly related to recognition time requirements, with faster required recognition time results in faster partition origination. At 155 msec, the recognition time requirement for the third experiment is the faster by 6 msec than the first and 17 msec than the second. Consequently, the rate of partition origination or PE usage is also proportionally higher in the third experiment compared to the previous two. At the same time, the number of PE's available is larger by 8 and 4 compared to the first and second experiments respectively. This counters the faster rate of PE usage and allows the DALI to scale to a vocabulary between the two experiments at 180 words.

The most significant observation remains the same as the previous experiments in that the architectural complexity of the DALI rises with algorithmic complexities, as new partitions originate and replace the ones that cross the TVS for real time selection. This ensures that the partitions selected in the real time (terminal partitions) are smaller than 25 words – the TVS, keeping the DALI's real time complexities simplified. The TVS is calibrated to the required recognition time, thus the expectation is that the DALI will maintain recognition time constant at 155 msec. We observed that the rate of increase in partition-PE numbers is faster compared to both previous experiments, because of a comparatively faster recognition time requirement. However, the DALI also leverages a larger number of PE's i.e. 16, which counters the faster rate of PE usage and allows the DALI to achieve a vocabulary size between previous experiments at 180 words.

8.5 Summary

We setup three experiments with the required recognition time and the number of PE's as the two experimental variables. The required recognition time is chosen to cover a broad range of human rates of speech ranging from 155 to 172 msec. And the number of PE's is chosen to examine the DALI's scalability with multi core scaling and increase from 8 up to 16. Figure 34 compares the experimental variables and the two dependent variables - Threshold Vocabulary Size (TVS) and the final vocabulary size achieved in each experiment set.

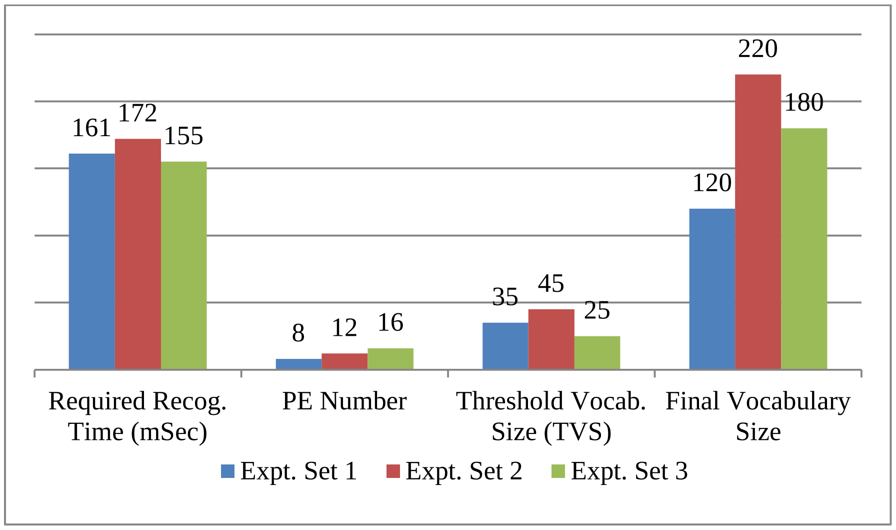


Figure 34: Experiment Sets Simulation Summary

The TVS value is calibrated to the required recognition times. Thus, its value at 35, 45 and 25 words is in proportion to the required recognition time value for the respective experiment i.e. 161, 172 and 155 msec. In all three sets we increase vocabulary in steps of 20 words and in response the DALI leverages increasing number of PE's to originate new partitions. We scale to vocabulary sizes of 120 words, 220 words and 180 words respectively in the three experiment sets. This seems surprising because the 3rd experiment set has a larger number of PE's than the 2nd and yet achieves a vocabulary size 40 words lower. The reason is that the rate at which partitions originate or the PE's are used by the DALI is proportional to the required recognition time. This is because new partitions originate in the DALI, when existing partitions cross the TVS, which is calibrated to the required recognition time. Thus, slower recognition time requirements result in a slower rate of PE usage. The recognition time requirement for the 2nd experiment set 2 is 17 msec slower than the 3rd, thus it is able to accommodate a larger vocabulary size, even over a smaller number of PE's. This indicates that the final achievable vocabulary size is dependent upon both the available architectural complexities and the required real time performance levels.

The most significant observation in the three experiments (Figures 30, 32, 33) is that the DALI's architectural complexities rise with algorithmic complexities, as new partitions, smaller than the

TVS originate to replace a partition that crosses the TVS, for real time selection. Consequently, the partitions selected by the DALI in real time (terminal partitions) are always smaller than the TVS. This ensures that the real time complexities of the DALI remain simplified despite an overall increase. Because the TVS is calibrated to required recognition times, we expect that this will allow the DALI to maintain recognition times constant at 161 msec, 172 msec and 155 msec respectively for the three experiments. In the next chapter we present the performance results of the DALI and analyze their conformity with this expectation.

9 Results and Analysis

We now summarize and analyze the results for the three experiment sets outlined in the previous chapter. These experiments examine the DALI's ability to maintain constant recognition time performance, as vocabulary sizes rise and an increasing number of PE's are leveraged. We also simulated the performance of an un-partitioned speech recognizer over the same vocabulary sizes as the DALI, to analyze comparative benefits of contextual partitioning. In the first three sub sections we analyze the results for each experiment set individually and subsequently we summarize our analysis.

9.1 Experiment Set 1

For experiment set 1, the required recognition time is defined as 161 msec, the Number of PE's as 8, and the simulations scaled to a final vocabulary size of 120 words. Figure 35 plots the results for experiments set 1.

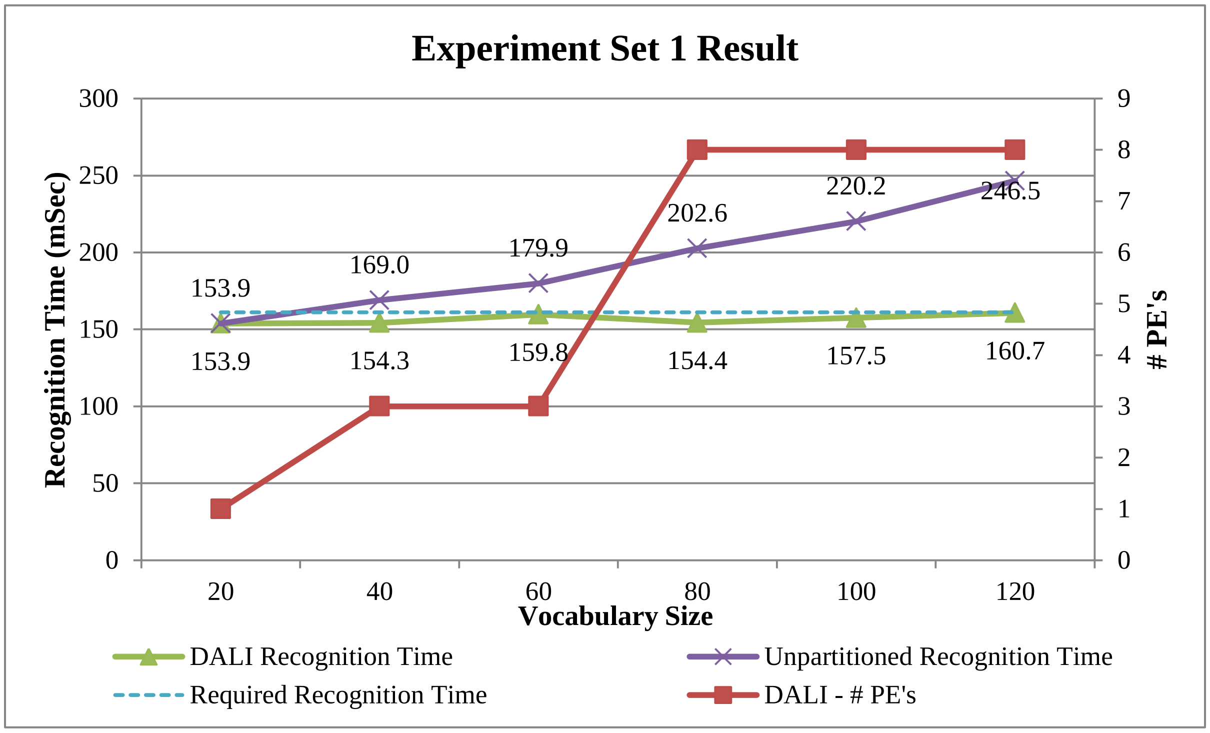


Figure 35: Experiment Set 1 Result

On the primary Y axis, DALI' recognition time is compared with un-partitioned recognizer and the required recognition time, which is depicted by dashed blue line running constant at 161 msec. On the secondary Y axis we plot the number of partition-PE pairs to analyze the impact of origination. A jump in this line corresponds to partition origination. The expectation is that the DALI’s recognition times will remain constant along the dashed line at 161 msec, indicating that it sucessfully achieves our performance goals.

DALI’s recognition time remains stable around 161 msec, however it is not strictly constant and varies slightly showing a small delta from 161 msec at most vocabulary sizes. However, with a minimum of 154 msec at 20 words, and a maximum of 160 msec at 120 words, the magnitude of the delta is negligible as it lies within 4 % of 161 msec. In comparison, the un-partitioned recognizers performance starts with at par with DALI, and worsens linearly with vocabulary size to end at 246 msec, an increase of almost 60 %. Interestingly, the slight variations in DALI’s performance and its delta from 161 msec, seems related to partition origination. In the absence of origination, recognition times slightly worsen, example: between 40-60 words by 5 msec, between 80-100 words by 3 msec and between 100-120 words by 3 msec. Contrarily, partition origination is accompanied with performance improvement, example: between 60-80 words recognition time falls by ~ 5 msec, or atleast no further degradation, example: between 20-40 words. The reason is that origination simplifies real time vocabularies keeping them below the 35 words – the TVS. Whereas in its absence, real time vocabularies increase because of integration. Significantly, prior to partition origination, example: at 60 words, DALI’s recognition times at 159.8 msec lies extremely close to 161 mSec – the required level. This means that in the absence of origination, at 80 words, DALI’s recognition time would have crossed 161 msec. This points to the effectiveness of MCP in regulating recognition times, and DALI’s ability to identify when origination is required.

The most significant observation in the figure is that the DALI's recognition time remains constant at 161 msec, as algorithmic and architectural complexities rise to 120 words and 8 PE’s respectively. This, despite the growing recognition time required by the unpartitioned speech recognizer. This confirms our expectation that the DALI, including the MCP supported by the dream phase, maintains constant real-time performance over growing algorithimic and architectural complexities.

9.2 Experiment Set 2

For experiment set 2, we defined the required recognition time and number of PE's at 172 msec and 12 respectively, and the simulations scaled to a final vocabulary size of 220 words. Figure 36 plots the results.

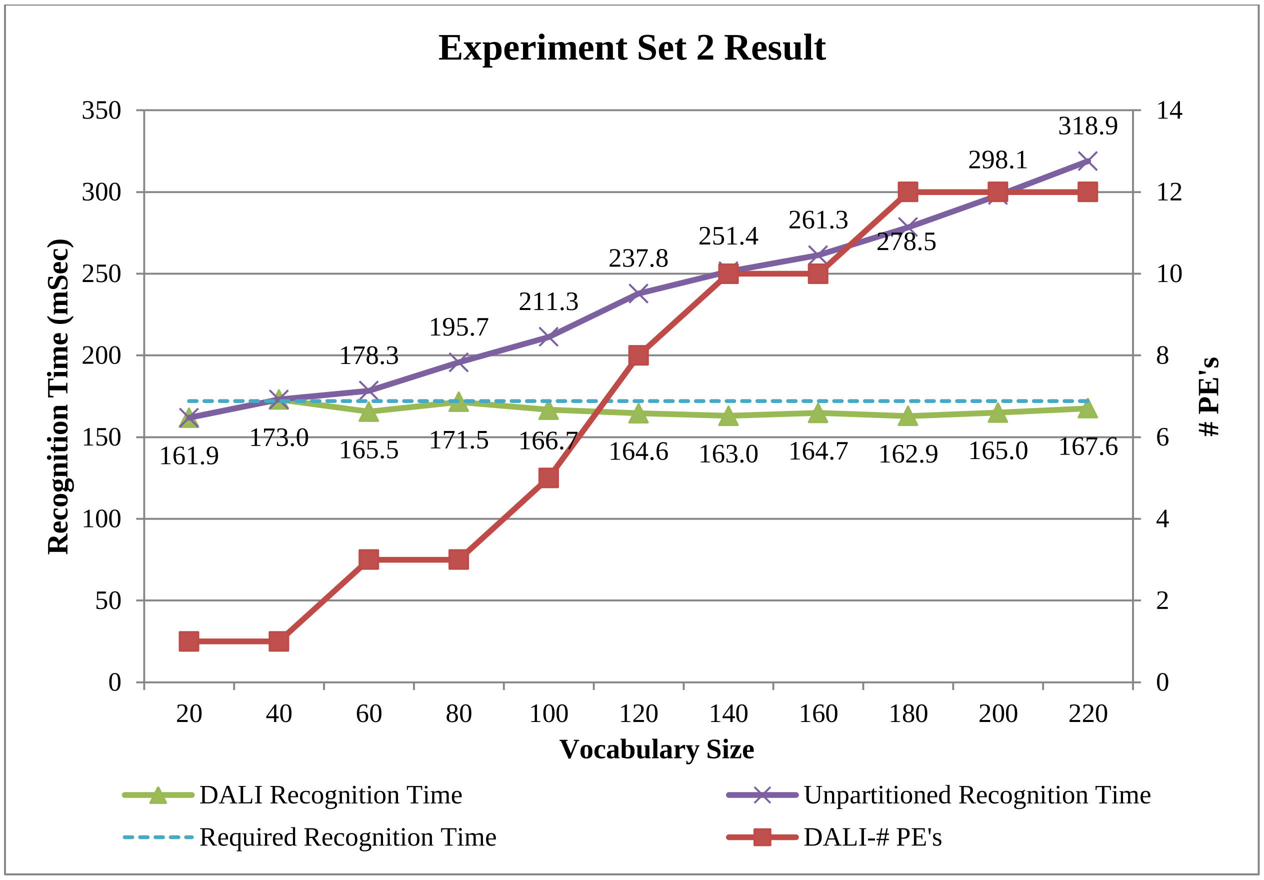


Figure 36: Experiment Set 2 Result

On the primary Y axis we compare the DALI's recognition time with the un-partitioned recognizers, and the required recognition time level, which is depicted by the dashed line running constant at 172 msec. On the secondary Y axis, the number of partition-PE pairs is plotted, and a jump in their number corresponds to partition origination. The expectation is that despite larger complexities compared to the previous experiment, the DALI's recognition times will remain constant along the dashed line at 172 msec. This will indicate that the DALI achieves the performance goals with scalability against algorithmic-architectural complexities.

Consistent with the previous set, DALI's recognition times remain stable around 172 msec, showing only slight variations, and a small delta from 172 msec at most vocabulary sizes. The magnitude of the delta is negligible, with a maximum delta of ~ 10 mSec at 20 words, meaning that all recognition times are within 5.8 % of 172 msec. Significantly, DALI never violates 172 msec, and actually performs better at most vocabulary sizes. In comparison, the un-partitioned recognizers performance degrades almost with vocabulary sizes, starting at par with DALI at 162 msec, and ending at 318 msec, a degradation of almost 100 % . Once again, the slight variations in DALI's recognition time, and their delta from 172 msec seems co-related with origination. Because, with an increase in PE numbers, the recognition times improves, example: between 40-60, 80-100 words. While when numbers are constant, we observe slight performance degradation, example: between 60-80 words, 20-40 words. Interestingly, at higher vocabulary sizes, the variation in recognition times with vocabulary size reduces. For example: concentrating on points where partitions originate, between 40-60 words, recognition time varies by ~7 msec, by 5 msec between 80-100 words and only ~1 msec between 120-140 words. The reason is that partition-PE's increase with vocabulary size. As a result, with increasing vocabularies, the user inputs are distributed over a larger set of partitions and an individual partition cannot dominate the DALI's performance. This is also why at initial vocabularies, example: between 60-80 words and 100-120 words, the DALI's performance is distinctly proximal to the required level–172 msec prior to origination, while this is not observed at later vocabulary sizes. From this it might seem that origination at latter vocabularies was un-needed. However, this is only because at later vocabularies, the proximity of individual partitions to the required recognition time – 172 msec is masked by other partitions.

The most significant observation in the figure is that the DALI's recognition time remains constant at 172 msec-the required level, as its algorithmic and architectural complexities increase well past the first experiment to 220 words and 12 PE's respectively. We observe that instead of degrading, the DALI's performance improves in stability as complexities grow. This, even as the recognition time for the un-partitioned recognizer continues to rise linearly with vocabulary size. These results confirm our expectation and prove that the DALI's ability to maintain constant performance is not limited to small algorithmic and architectural complexity levels.

9.3 Experiment Set 3

For experiment set 3, we defined the required recognition time and number of PE's as 155 msec and 16 respectively. The simulations scaled to a final vocabulary size of 180 words. Figure 37 plots results for experiment set 3.

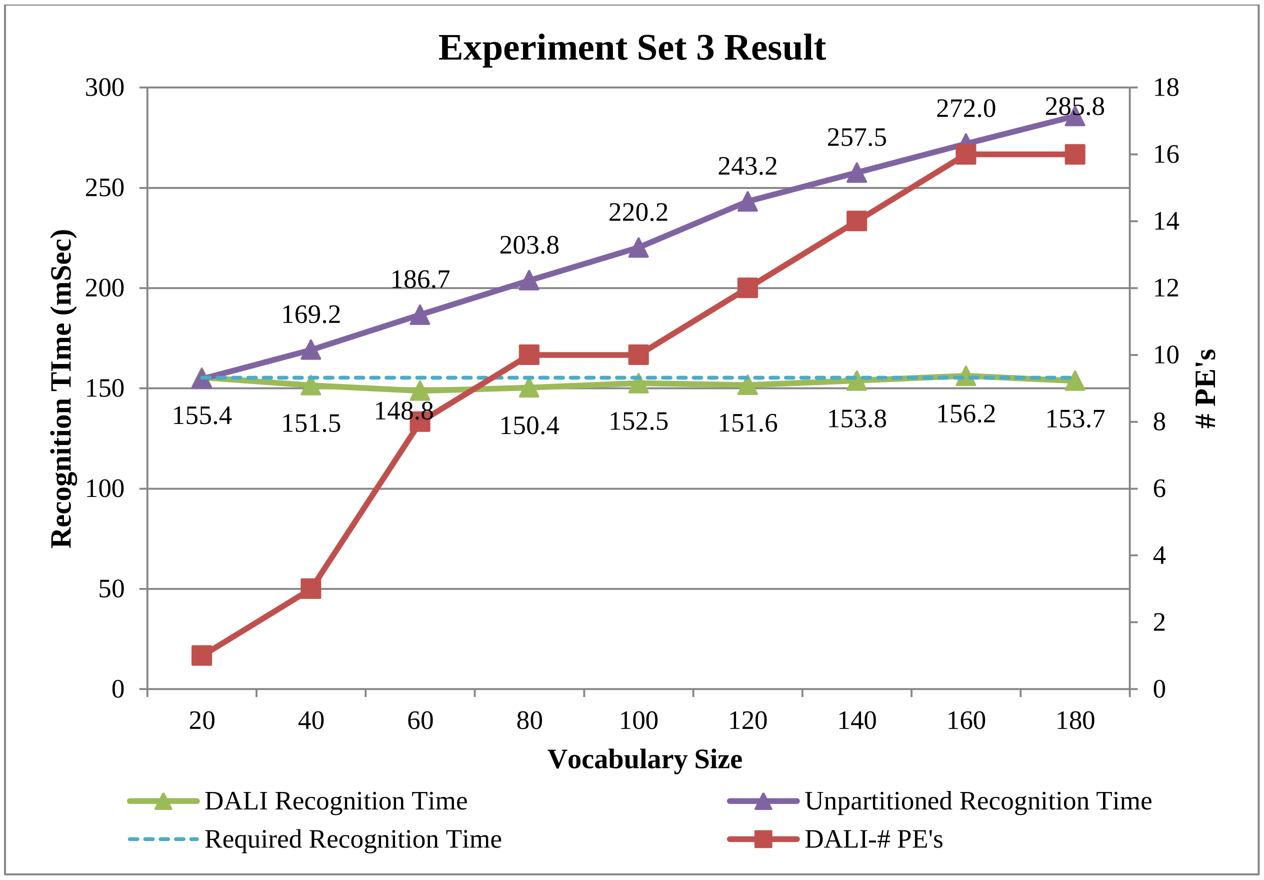


Figure 37: Experiment Set 3 Result

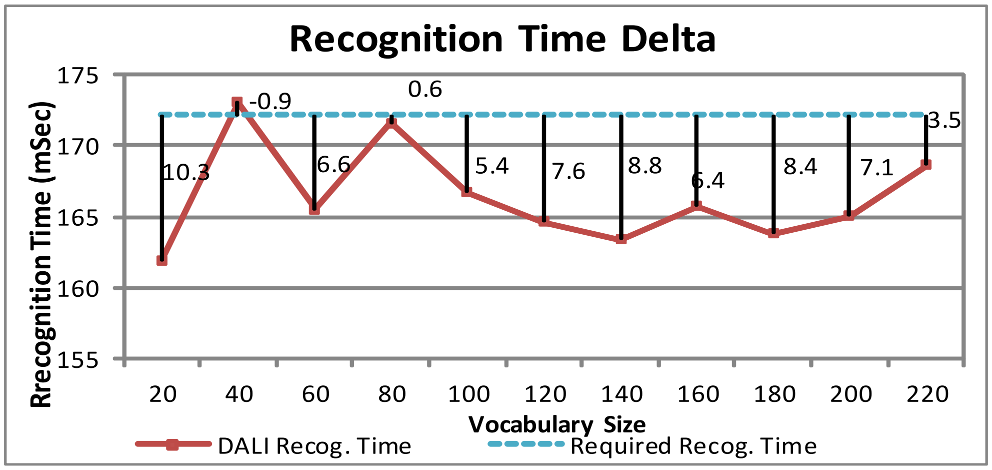
On the primary Y axis we compare DALI's recognition times with un-partitioned recognizer and the required recognition time level, which is depicted by the dashed line running constant at 155 msec. The secondary Y axis plots the number of partition-PE pairs for DALI at each vocabulary size, with a jump corresponding to origination. Consistent with previous experiments, the expectation is that the DALI will maintain recognition times constant along the dashed line at 155 msec, despite scaling to larger complexities than the previous experiment sets. This would confirm that the DALI achieves our performance goals without a loss of scalability against algorithmic-architectural complexities.

Consistent with the previous experiments, the DALI's recognition times remain stable around 155 msec. They are not strictly constant, as they vary slightly and maintain a small delta from 155 msec at most vocabularies. However, the magnitude of delta is negligible, with all values lying within 4.8 % of 155 msec, also DALI never violates 155 mSec, rather performs better at most vocabularies. In comparison, the un-partitioned recognizer's performance worsens almost linearly with vocabulary sizes, starting at par with DALI and ending at 285.8 msec, a degradation of almost 80 %. Other observations are consistent with previous experiments and are discussed only briefly here. The slight variation in the DALI's recognition time and its delta from 155 msec is correlated with origination, as recognition times improve when PE numbers rise, and increase when they remain constant. Also, DALI's performance becomes increasingly stable as we approach higher vocabulary sizes and partition-PE pair numbers, because the impact of an individual partition on the average recognition time reduces. This is also why at small vocabulary sizes and partition-PE numbers, prior to an origination we see a distinct proximity between the DALI's recognition time and 155 msec-the required level. Whereas, with latter vocabularies this is not the case, as the proximity of individual partitions to 155 msec is hidden and effectively averaged out by other partitions.

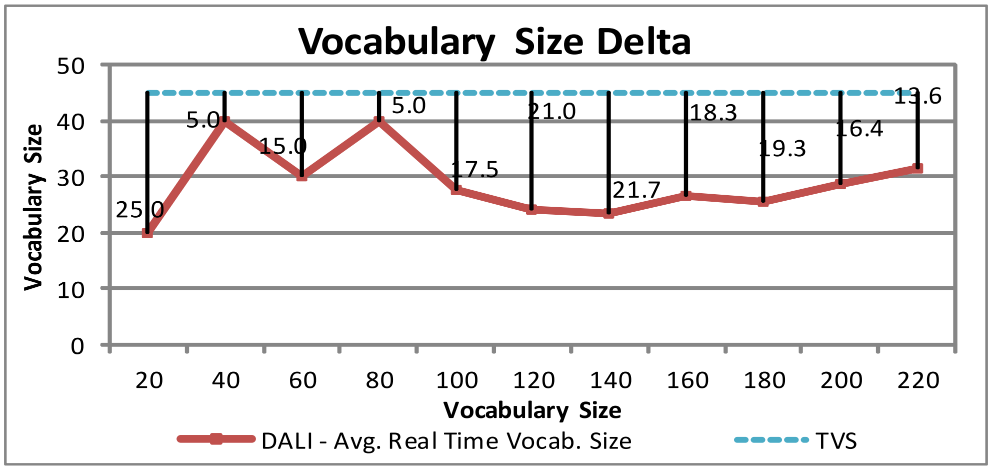
The most significant observation remains consistent with previous experiments in that the DALI maintains recognition time constant at the required level of 155 msec as its algorithmic and architectural complexities rise to 180 words and 16 PE's respectively. This means, that despite scaling to 60 words higher and using double the number of PE's than the first experiment, there is no performance degradation. Rather we observe that the DALI's performance becomes increasingly stable. This, even as the un-partitioned recognizer's performance rises against vocabulary size. The consistency in the results with the previous experiments proves that the DALI's ability to maintain constant performance scales almost ideally over the investigated algorithmic and architectural complexities.

9.4 Optimization with Heterogeneity

In all the three experiment sets the DALI performs slightly better than the required recognition time. The delta from the required recognition time is small, lying within 4-6 % and does not impact the user negatively. However, performance better than the required real time level is wasted. This presents an optimization opportunity that can improve the power consumption of the DALI. We observed that the recognition time delta is correlated with partition origination. The reason is that origination simplifies partition vocabularies, pushing them below the TVS level, which is calibrated to the required recognition time. As a result the DALI's recognition times would also remain below the required level. To confirm this analysis, Figure 38 compares the delta between the required and the DALI's recognition time, and between the TVS and the DALI's average real time vocabulary size for experiment set 2.



(a)



(b)

Figure 38: Delta between (a) Recognition times (Required & the DALI), (b) Vocabulary Size (TVS & the DALI)

In Figure 37(a), the dashed blue line depicts the required recognition time of 172 msec, and the drop lines to DALI's recognition times depict the delta between the two. Thus, the space between the blue line and the DALI's recognition time can be considered to be wasted. And, in Figure 37 (b), the dashed blue line represents the TVS of 45 words, and the drop lines, the delta with the DALI's average real time vocabulary size. As expected, DALI's recognition time shows a striking similarity to DALI's average real time vocabulary sizes. Also, recognition time delta and vocabulary size delta seem to be proportional, with the points of minimum/maximum delta coinciding. Example: Minimum of -0.9 and 0.6 msec corresponding to 5 Words, at 40 and 80 words, and maximum of 10 msec corresponding to 25 words, at vocabulary size of 20 words. This proves that the recognition time delta is a result of the DALI's real time vocabulary size remaining slightly lower than the TVS. The reason is that all partitions, irrespective of their vocabulary size are allocated to a homogeneous set of PE's. As a result, partitions with vocabulary size below the TVS are over-provisioned with PE's that are stronger than required to maintain the ideal straight line. A solution to this is the incorporation of heterogeneous PE's on the DALI. This would enable partition-PE allocation algorithms that optimally match the computational power of a PE to the partitions vocabulary size, and enable the DALI to approach the ideal, flat recognition time. In addition, this would optimize the DALI's power consumption, as simpler PE's will be allocated to proportionally smaller partitions. These will run at a lower frequencies and voltages while still maintaining real time performance, and thus consume lower power compared to homogeneous PE's. Next, we summarize the results for the three experiments and our analysis.

9.5 Summary

Figure 39 summarizes the DALI's performance in the three experiment sets. Previously, in chapter 8, we established that the number of PE's in the DALI's are coupled with increase in vocabulary size. Thus, the DALI's recognition times are plotted against both vocabulary size and corresponding number of PE's. For the vocabulary size that is not reached in an experiment set (140 onwards for experiment set 1 and 180 onwards for experiment set 3), the # PE's is marked as 'X' to represent a non-applicable value.

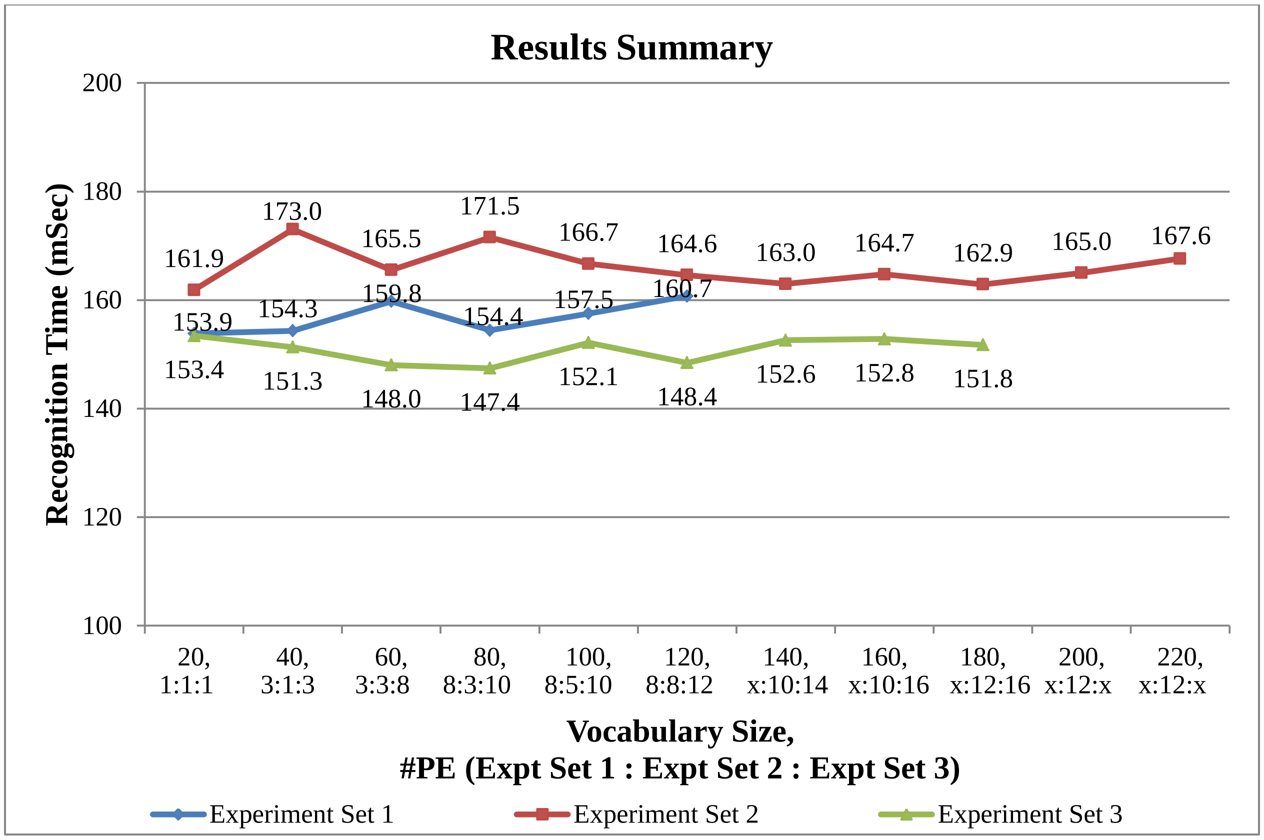


Figure 39: DALI Results Summary

Consistently, in all the three experiments, the DALI maintains recognition times constant at the required level – 161, 172 and 155 msec, respectively. This, even as, the DALI's algorithmic and architectural complexities grow larger over the experiments, as the final vocabulary size rises from 120 to 220 words and the number of PE's from 8 to 16. While, the DALI's performance is not strictly flat, as envisioned in Figure 1, the variations are negligible as they lie within 4-6 % of the required recognition time. Also, they have no user impact as the DALI's recognition times remain slightly better than the required level. Furthermore, we analyzed that even these variations can be removed with the incorporation of heterogeneity on the DALI.

We can thus reasonably conclude that in all the three experiments, the DALI successfully achieved the performance goals defined in Figure 1, proving that it can maintain constant performance with almost ideal scalability against algorithmic and architectural complexities.

These results mark a paradigm shift from traditional architectures that cannot scale beyond 4-6 PE's and form a reasonable basis for future work. In the next chapter we conclude our research effort and identify some key challenges for future research.

10 Conclusions and Future Work

The primary major contribution of our research is to show that the DALI, utilizing Computational Dreaming to support the Malleable Contextual Partitioning approach, can maintain constant real time performance against rising algorithmic and architectural complexities for database dependent applications. In achieving this, the secondary major contributions are twofold: First, we propose a method for malleable contextual partitioning (MCP) that personalizes contextual partitions to user behavior. Second, we conceptualize and simulate a novel architectural framework-the Dream Architecture for Lateral Intelligence (DALI) to demonstrate the MCP approach. We target speech recognition as an example application, and setup three experiments. These experiments cover a broad range of recognition time requirements corresponding to human rates of speech and use an increasing number of PE's in accordance with the trend of multi-core scaling. The DALI consistently maintains recognition time constant at the required level for each experiment - 161, 172 and 155 msec respectively. This, even as over the three experiments, the DALI scales to increasing algorithmic and architectural complexities, as vocabulary sizes up to 220 words are achieved while leveraging up to 16 PE's. These results prove that the DALI in support of the MCP approach can maintain constant real-time performance with ideal scalability over the range of algorithmic complexity and architectural sizes investigated. Thus, we successfully achieve our primary contribution while illustrating how the MCP algorithm supported by the DALI's architectural conceptualization is consistent with a Computational Dreaming approach.

10.1 Future Work

These results provide a reasonable basis for conducting further research on the MCP approach and the DALI. We identify the following major challenges for future research:

- Simulation and Optimization of the DALI's power consumption
- Development of Partition Origination, Retirement and Scheduling algorithms
- Conceptualization and Development of a Partition Relationship algorithm

These are discussed in further detail here.

DALI's Power Simulation and Optimization

For the DALI to provide an alternative personal computing architecture, its power consumption both in the real time and in the dream phase must remain bounded, despite an overall increase in complexities. For this analysis, it is critical that future research simulate the DALI's power consumption to identify inefficient algorithmic-architectural components for optimization. The first step is the development of a power simulator for the DALI. A key trade-off that must be considered is between the accuracies and setup, simulation times. We suggest that initially the researchers investigate simulation at high levels of abstraction, trading off accuracy in the favor of simulation times, which is feasible considering the early design phase of the DALI. Researchers can derive inspiration from existing research in high level power simulation with toolsets such as MESH [108]. Also, the integration of power annotations into the DALI simulator, as developed in this research, can provide a viable alternative.

Analysis of the power simulation results will identify the architectural, algorithmic components that require optimization. Between the two phases i.e. the real time and the dream phase, we expect that the dream phase will be a prime candidate for power optimization. The reason being that in the real time the DALI uses simplified complexities despite an overall increase. Contrarily, an increasing number of PE's and algorithmic complexities are exercised in MISD parallelism when dreaming. While the MISD parallelism is crucial in order to ensure bounded times for the dream phase, as a larger number of PE's operate simultaneously, the DALI will eventually violate the Thermal Dissipation Power (TDP) limit. In solution, we suggest that future work investigate the division of the dream phase into multiple cycles. Each cycle would exercise only a subset of the partition-PE's in MISD parallelism, and thus the dream phase would take multiple cycles to cover all the PE's. The size and the contents (i.e. the partition-PE's) of the subset can be chosen to develop an optimal trade-off between power and time consumption. We believe that these power constraints are consistent with the human brain, because of which all memories cannot be replayed at once in the dream sleep. This might be one reason behind the division of the dream sleep into multiple alternating cycles of REM and NREM sleep, instead of a contiguous cycle.

Partition Origination, Retirement and Scheduling Algorithms

In the DALI, as conceptualized, we have proposed approaches to, but stopped short of developing three key algorithms: the partition scheduling algorithm, which is used by the Context Activation Processor (CAP), and the partition origination, retirement algorithms, which are sub components of the MCP algorithm (Figure 12). The approaches proposed for the three algorithms are: the predictive scheduling approach (in Chapter 6), spatio-temporal approach towards origination (Chapter 4) and the step-wise retirement approach (Chapter 4). These approaches provide a solid conceptual foundation for the three algorithms, thus, their development and evaluation presents an interesting immediate challenge towards completing the DALI, as conceptualized. Key evaluation metrics for the three algorithms are the accuracy (in scheduling, retirement and origination respectively), and the completion times.

However, accuracy in these algorithms comes with a closer user personalization, and will be coupled with longer completion times. This is because personalization entails an extensive analysis of the user behavior, and at times, enough contextual information will simply not be available for making an accurate decision, especially in the initial phase, when the user behavior is still being learnt. Observing this, we suggest that researchers take a probabilistic approach in developing these algorithms, such that an approximately accurate decision is made within bounded time. This probabilistic approach would trade-off accuracies in the favor of completion times. However the dream phase can be leveraged to analyze the resultant inaccuracies and improve the probabilities for future decisions. This is consistent with the human brain that often makes approximate decisions either because of limited information or to ensure timely response, and only gradually improves decision accuracies as it learns with age.

Inter-Partition Relationship algorithm

Currently, the MCP approach includes three fundamental partition characteristics-origination, integration and retirement. For a closer personalization to the user, researchers must identify, conceptualize additional partition characteristics, and evaluate them algorithmically in the DALI. A key characteristic that merits further investigation is the inter-partition relationships in the spatio-temporal hierarchy. Currently, the only relationship we identify and leverage is that of a parent and a child. Another apparent relationship is that of *companion partitions*, between those that are selected consecutively or close together in the real time. The selection of one of these partitions will provide a cue for the selection of its companion. This cue can be leveraged by the partition scheduling algorithm in two ways: First, it can reinforce the received contextual inputs to select a partition with higher accuracies. Second, it can be used to predictively select a partition, much before the actual inputs are received, significantly lowering the scheduling times. The relationships themselves can be weighted with strength values, which can be increased or decreased in the dream phase analysis. The weights can be used to improve prediction accuracies, by heeding to only strong relationships. As additional relationships are conceptualized and implemented, the spatio-temporal hierarchy will become densely connected, enabling accurate, fast predictive selection. This will be a step closer to the operation of the brain, where prediction seems to be crucial in reducing memory retrieval search space, and enabling pre-emptive action to external stimulus which leaves enough time to correct inaccurate decisions.

Apart from these challenges, the implementation of the MCP and Computational Dreaming approach to other applications must also be investigated. Candidates, as discussed in the background chapter, are the set of database dependent applications. A common characteristic of these applications is that they can produce multiple outputs, even to the same input, and the correct output pertains to the user’s context. We recommend that researchers focus on multiplexing a second recognition algorithm alongside speech on the DALI, and evaluate if performance goals (Figure 1) can be achieved with both. This will pose interesting challenges for both the MCP approach and the DALI, including mechanisms for inter-application partition, architectural resource sharing and inter application partition relationships. The solutions to these will evolve the DALI into a platform for multi-modal recognition, in support of an application agnostic and densely related spatio-temporal partition hierarchy. This is inspired from the brain where a myriad set of recognition tasks share the cortical real estate, often with flexible boundaries, and supplement each other to improve recognition accuracy. For example: lip reading often complements speech recognition. A potential candidate for multiplexing with speech is gesture recognition, as multi-modal recognition with speech and gestures has already shown benefits in smartphones.

Research into these challenges will pose further questions, providing impetus to closer investigation of the structural, operational characteristics of the human brain for integration into computer architectures. The flow of inspiration need not be one dimensional i.e. from the brain towards computers. Rather, as brain inspired computer architectures hit computational limits, research into their solutions will further the understanding of the brains operation and structure. In this way, we hope that our thesis motivates a larger body of work with the overall goal of bridging human and computer intelligence.

References

[1] Uchechukwu, A., Keqiu, L., and Yanming, S.: 'Improving cloud computing energy efficiency': 'Book Improving cloud computing energy efficiency' (2012, edn.), pp. 53-58

[2] http://testyourvocab.com/: 'Test your vocab', 2013

[3] Merkle, R.C.: 'Energy limits to the computational power of the human brain', Foresight Update, 1989, 6

[4] Stickgold, R., Hobson, J.A., Fosse, R., and Fosse, M.: 'Sleep, learning, and dreams: off-line memory reprocessing', Science, 2001, 294, (5544), pp. 1052-1057

[5] Zhang, Q.: 'A computational account of dreaming: learning and memory consolidation', Cognitive Systems Research, 2009, 10, (2), pp. 91-101

[6] Campbell, A., and Choudhury, T.: 'From Smart to Cognitive Phones', Pervasive Computing, IEEE, 2012, 11, (3), pp. 7-11

[7] Lumsdaine, A., Gregor, D., Hendrickson, B., and Berry, J.: 'CHALLENGES IN PARALLEL GRAPH PROCESSING', Parallel Processing Letters, 2007, 17, (01), pp. 5-20

[8] J. L. Hennessy, D.A.P.: 'Computer Architecture, Fourth Edition: A Quantitative Approach', Morgan Kaufmann Publishers Inc., 2006

[9] Taylor, M.B.: 'Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse'. Proc. Proceedings of the 49th Annual Design Automation Conference, San Francisco, California2012 pp. Pages

[10] Herlihy, M., and Luchangco, V.: 'Distributed computing and the multicore revolution', SIGACT News, 2008, 39, (1), pp. 62-72

[11] Cattell, R., and Parker, A.: 'Challenges for brain emulation: why is it so difficult', Nat. Intell, 2012, 1, (3), pp. 17-31

[12] Jeff Hawkins, S.B.: 'On Intelligence', 2005

[13] Tulving, E.: 'Episodic and Semantic Memory1', Organization of memory, 1972, pp. 381-402

[14] Tulving, E.: 'Episodic memory: from mind to brain', Annual review of psychology, 2002, 53, pp. 1-25

[15] Dubuc, B.: 'The Brain from Top to Bottom', McGill University, 2002

[16] Cangelosi, A., and Harnad, S.: 'The adaptive advantage of symbolic theft over sensorimotor toil: Grounding language in perceptual categories': 'Book The adaptive advantage of symbolic theft over sensorimotor toil: Grounding language in perceptual categories' (2001, edn.), pp. 117-142

[17] Harnad, S.R.: 'Categorical perception: The groundwork of cognition' (Cambridge University Press, 1990. 1990)

[18] Hofstadter, D.R.: 'On seeing A's and seeing As', Stanford Hum. Rev., 1995, 4, (2), pp. 109-121

[19] Numenta: 'Hierarchical Temporal Memory including HTM Cortical Learning Algorithms', 2011

[20] Walker, M.P.: 'A refined model of sleep and the time course of memory formation', Behavioral and brain sciences, 2005, 28, (1), pp. 51-63

[21] Maquet, P.: 'The role of sleep in learning and memory', Science, 2001, 294, (5544), pp. 1048-1052

[22] Peigneux, P., Laureys, S., Delbeuck, X., and Maquet, P.: 'Sleeping brain, learning brain. The role of sleep for memory systems', NeuroReport, 2001, 12, (18), pp. A111-A124

[23] Rock., A.: 'The Mind at Night: The New Science of How and Why We Dream', Basic Books,, 2005

[24] Baylor, G.W., and Cavallero, C.: 'Memory sources associated with REM and NREM dream reports throughout the night: a new look at the data', Sleep, 2001, 24, (2), pp. 165

[25] Battaglia, F.P., Sutherland, G.R., and McNaughton, B.L.: 'Hippocampal sharp wave bursts coincide with neocortical "up-state" transitions', Learning & Memory, 2004, 11, (6), pp. 697-704

[26] Girardeau, G., Benchenane, K., Wiener, S.I., Buzsáki, G., and Zugaro, M.B.: 'Selective suppression of hippocampal ripples impairs spatial memory', Nature neuroscience, 2009, 12, (10), pp. 1222-1223

[27] Ego-Stengel, V., and Wilson, M.A.: 'Disruption of ripple-associated hippocampal activity during rest impairs spatial learning in the rat', Hippocampus, 2010, 20, (1), pp. 1-10

[28] Buzsáki, G.: 'The hippocampo-neocortical dialogue', Cerebral Cortex, 1996, 6, (2), pp. 81-92

[29] Stickgold, R.: 'Sleep: off-line memory reprocessing', Trends in Cognitive Sciences, 1998, 2, (12), pp. 484-492

[30] Káli, S., and Dayan, P.: 'Off-line replay maintains declarative memories in a model of hippocampal-neocortical interactions', Nature neuroscience, 2004, 7, (3), pp. 286-294

[31] Weiser, M.: 'Some computer science issues in ubiquitous computing', Communications of the ACM, 1993, 36, (7), pp. 75-84

[32] Lippmann, R.P.: 'Speech recognition by machines and humans', Speech communication, 1997, 22, (1), pp. 1-15

[33] E. Alpaydin, F.G.: 'Comparison of Statistical and Neural Classifiers and Their Applcations to Optical Character Recognition and Speech Classification', Neural network systems techniques and applications, 1996, pp. 61-88

[34] García-Moral, A., Solera-Ureña, R., Peláez-Moreno, C., and Díaz-de-María, F.: 'Hybrid Models for Automatic Speech Recognition: A Comparison of Classical ANN and Kernel Based Methods': 'Advances in Nonlinear Speech Processing' (Springer Berlin Heidelberg, 2007), pp. 152-160

[35] 'Configuration of Hidden Markov Models': 'Markov Models for Pattern Recognition' (Springer Berlin Heidelberg, 2008), pp. 127-136

[36] Rabiner, L.: 'A tutorial on hidden Markov models and selected applications in speech recognition', Proceedings of the IEEE, 1989, 77, (2), pp. 257-286

[37] Kent, C.G.: 'Personalized Computer Architecture as Contextual Partitioning for Speech Recognition', Virginia Polytechnic Institute and State University, 2009

[38] Foote, J.T., Hochberg, M.M., Athanas, P.M., Smith, A.T., Wazlowski, M.E., and Silverman, H.F.: 'Distributed hidden Markov model training on loosely-coupled multiprocessor networks': 'Book Distributed hidden Markov model training on loosely-coupled multiprocessor networks' (1992, edn.), pp. 569-572 vol.564

[39] Popescu, V., Burileanu, C., Rafaila, M., and Calimanescu, R.: 'Parallel training algorithms for continuous speech recognition, implemented in a message passing framework', Proc. Eusipco, Florence, 2006

[40] Buthpitiya, S., Lane, I., and Chong, J.: 'A parallel implementation of Viterbi training for acoustic models using graphics processing units': 'Book A parallel implementation of Viterbi training for acoustic models using graphics processing units' (IEEE, 2012, edn.), pp. 1-10

[41] Willie Walker, P.L., Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, Joe Woelfel: ‘Sphinx-4: A Flexible Open Source Framework for Speech Recognition’
[42] Chong, J., Friedland, G., Janin, A., Morgan, N., and Oei, C.: ‘Opportunities and challenges of parallelizing speech recognition’. Proc. Proceedings of the 2nd USENIX conference on Hot topics in parallelism, Berkeley, CA2010 pp. Pages
[43] Kartik K. Agaram , S.W.K., Doug Burger: ‘Characterizing the SPHINX Speech Recognition System’, 2001
[44] Flynn, M.: ‘Very high-speed computing systems’, Proceedings of the IEEE, 1966, 54, (12), pp. 1901-1909
[45] Halaas, A., Svingen, B., Nedland, M., Saetrom, P., Snove, O., and Birkeland, O.R.: ‘A recursive MISD architecture for pattern matching’, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2004, 12, (7), pp. 727-734
[46] Arora, M.: ‘The Architecture and Evolution of CPU-GPU Systems for General Purpose Computing’, Research survey, University of California, San Diego, 2012
[47] Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., and Phillips, J.C.: ‘GPU Computing’, Proceedings of the IEEE, 2008, 96, (5), pp. 879-899
[48] Kindratenko, V.V., Enos, J.J., Guochun, S., Showerman, M.T., Arnold, G.W., Stone, J.E., Phillips, J.C., and Wen-Mei, H.: ‘GPU clusters for high-performance computing’: ‘Book GPU clusters for high-performance computing’ (2009, edn.), pp. 1-8
[49] Scogland, T.R.W., Lin, H., and Feng, W.: ‘A first look at integrated GPUs for green high-performance computing’, Comput Sci Res Dev, 2010, 25, (3-4), pp. 125-134
[50] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., and Skadron, K.: ‘A performance study of general-purpose applications on graphics processors using CUDA’, J. Parallel Distrib. Comput., 2008, 68, (10), pp. 1370-1380
[51] Gregg, C., and Hazelwood, K.: ‘Where is the data? Why you cannot debate CPU vs. GPU performance without the answer’. Proc. Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software2011 pp. Pages
[52] Lustig, D., and Martonosi, M.: ‘Reducing GPU offload latency via fine-grained CPU-GPU synchronization’: ‘Book Reducing GPU offload latency via fine-grained CPU-GPU synchronization’ (2013, edn.), pp. 354-365
[53] Borkar, S.: ‘Thousand core chips: a technology perspective’. Proc. Proceedings of the 44th annual Design Automation Conference, San Diego, California2007 pp. Pages
[54] Amdahl, G.M.: ‘Validity of the single processor approach to achieving large scale computing capabilities’. Proc. Proceedings of the April 18-20, 1967, spring joint computer conference, Atlantic City, New Jersey1967 pp. Pages
[55] Pankratius, V., Schaefer, C., Jannesari, A., and Tichy, W.F.: ‘Software engineering for multicore systems: an experience report’. Proc. Proceedings of the 1st international workshop on Multicore software engineering, Leipzig, Germany2008 pp. Pages
[56] Kumar, R., Zyuban, V., and Tullsen, D.M.: ‘Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling’: ‘Book Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling’ (IEEE, 2005, edn.), pp. 408-419
[57] ‘System Drivers Report, 2011’, International Technology Roadmap for Semiconductors
[58] Ravishankar, M.: ‘Parallel implementation of fast beam search for speaker-independent continuous speech recognition’, Computer Science & Automation, Indian Institute of Science, Bangalore, India, 1993

[59] Agaram, K., Keckler, S.W., and Burger, D.: 'A characterization of speech recognition on modern computer systems': 'Book A characterization of speech recognition on modern computer systems' (IEEE, 2001, edn.), pp. 45-53
[60] Paulson, L.D.: 'Speech recognition moves from software to hardware', Computer, 2006, 39, (11), pp. 15-18
[61] Nedevschi, S., Patra, R.K., and Brewer, E.A.: 'Hardware speech recognition for user interfaces in low cost, low power devices': 'Book Hardware speech recognition for user interfaces in low cost, low power devices' (IEEE, 2005, edn.), pp. 684-689
[62] Krishna, R., Mahlke, S., and Austin, T.: 'Architectural optimizations for low-power, real-time speech recognition': 'Book Architectural optimizations for low-power, real-time speech recognition' (ACM, 2003, edn.), pp. 220-231
[63] Arora, M.: 'The Architecture and Evolution of CPU-GPU Systems for General Purpose Computing', By University of California, San Diago
[64] Dixon, P.R., Caseiro, D.A., Oonishi, T., and Furui, S.: 'The Titech large vocabulary WFST speech recognition system': 'Book The Titech large vocabulary WFST speech recognition system' (IEEE, 2007, edn.), pp. 443-448
[65] Chong, J., Gonina, E., Yi, Y., and Keutzer, K.: 'A fully data parallel WFST-based large vocabulary continuous speech recognition on a graphics processing unit': 'Book A fully data parallel WFST-based large vocabulary continuous speech recognition on a graphics processing unit' (2009, edn.), pp. 1183
[66] Kim, J., Chong, J., and Lane, I.: 'Efficient On-The-Fly Hypothesis Rescoring in a Hybrid GPU/CPU-based Large Vocabulary Continuous Speech Recognition Engine'
[67] http://singularityhub.com/: 'Switch to tablet and smartphone drives striking sales declines in PCs', 2013
[68] Lin, C.-T., Ko, L.-W., Chang, C.-J., Wang, Y.-T., Chung, C.-H., Yang, F.-S., Duann, J.-R., Jung, T.-P., and Chiou, J.-C.: 'Wearable and Wireless Brain-Computer Interface and Its Applications': 'Foundations of Augmented Cognition. Neuroergonomics and Operational Neuroscience' (Springer Berlin Heidelberg, 2009), pp. 741-748
[69] Campbell, A., Choudhury, T., Hu, S., Lu, H., Mukerjee, M.K., Rabbi, M., and Raizada, R.D.S.: 'NeuroPhone: brain-mobile phone interface using a wireless EEG headset'. Proc. Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds, New Delhi, India2010 pp. Pages
[70] 'Qualcomm Snapdragon 600 vs Apple A6 (APL0598) ', 2013
[71] 'Qualcomm Snapdragon 600 vs Nvidia Tegra 4', 2013
[72] Kumar, R., Tullsen, D.M., and Jouppi, N.P.: 'Core architecture optimization for heterogeneous chip multiprocessors': 'Book Core architecture optimization for heterogeneous chip multiprocessors' (ACM, 2006, edn.), pp. 23-32
[73] Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P., and Tullsen, D.M.: 'Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction': 'Book Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction' (IEEE, 2003, edn.), pp. 81-92
[74] Benini, L., and De Micheli, G.: 'Networks on chips: A new SoC paradigm', Computer, 2002, 35, (1), pp. 70-78
[75] Dally, W.J., and Towles, B.: 'Route packets, not wires: On-chip interconnection networks': 'Book Route packets, not wires: On-chip interconnection networks' (IEEE, 2001, edn.), pp. 684-689

[76] Angiolini, F., Meloni, P., Carta, S., Benini, L., and Raffo, L.: 'Contrasting a NoC and a traditional interconnect fabric with layout awareness': 'Book Contrasting a NoC and a traditional interconnect fabric with layout awareness' (European Design and Automation Association, 2006, edn.), pp. 124-129

[77] Benini, L.D.M.G.: 'Networks on chips : technology and tools' (Elsevier Morgan Kaufmann Publishers, 2006. 2006)

[78] Arteris: 'A comparison of Network-on-Chip and Busses- A White Paper'

[79] Arteris: 'From "Bus" and "Crossbar" to "Network-On-Chip"'

[80] De Micheli, G., Seiculescu, C., Murali, S., Benini, L., Angiolini, F., and Pullini, A.: 'Networks on chips: From research to products': 'Book Networks on chips: From research to products' (IEEE, 2010, edn.), pp. 300-305

[81] Lane, N.D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., and Campbell, A.T.: 'A survey of mobile phone sensing', Communications Magazine, IEEE, 2010, 48, (9), pp. 140-150

[82] Miluzzo, E., Lane, N.D., Eisenman, S.B., and Campbell, A.T.: 'CenceMe–injecting sensing presence into social networking applications': 'Smart Sensing and Context' (Springer, 2007), pp. 1-28

[83] Lane, N.D., Mohammod, M., Lin, M., Yang, X., Lu, H., Ali, S., Doryab, A., Berke, E., Choudhury, T., and Campbell, A.: 'BeWell: A smartphone application to monitor, model and promote wellbeing': 'Book BeWell: A smartphone application to monitor, model and promote wellbeing' (2011, edn.), pp.

[84] You, C.-W., Montes-de-Oca, M., Bao, T.J., Lane, N.D., Lu, H., Cardone, G., Torresani, L., and Campbell, A.T.: 'CarSafe: a driver safety app that detects dangerous driving behavior using dual-cameras on smartphones'. Proc. Proceedings of the 2012 ACM Conference on Ubiquitous Computing, Pittsburgh, Pennsylvania2012 pp. Pages

[85] Choudhury, T., Consolvo, S., Harrison, B., Hightower, J., LaMarca, A., LeGrand, L., Rahimi, A., Rea, A., Bordello, G., and Hemingway, B.: 'The mobile sensing platform: An embedded activity recognition system', Pervasive Computing, IEEE, 2008, 7, (2), pp. 32-41

[86] Liao, L., Fox, D., and Kautz, H.: 'Extracting Places and Activities from GPS Traces Using Hierarchical Conditional Random Fields', Int. J. Rob. Res., 2007, 26, (1), pp. 119-134

[87] Schmidt, A., Aidoo, K.A., Takaluoma, A., Tuomela, U., Van Laerhoven, K., and Van de Velde, W.: 'Advanced interaction in context': 'Book Advanced interaction in context' (Springer, 1999, edn.), pp. 89-101

[88] Dey, A.K.: 'Understanding and using context', Personal and ubiquitous computing, 2001, 5, (1), pp. 4-7

[89] Kumar, K., and Yung-Hsiang, L.: 'Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?', Computer, 2010, 43, (4), pp. 51-56

[90] Begley, S.: 'How the brain rewires itself', TIME, 2009

[91] Kornell, N., Hays, M.J., and Bjork, R.A.: 'Unsuccessful retrieval attempts enhance subsequent learning', Journal of Experimental Psychology: Learning, Memory, and Cognition, 2009, 35, (4), pp. 989

[92] Hacker, M.: 'Context-aware speech recognition in a robot navigation scenario': 'Book Context-aware speech recognition in a robot navigation scenario' (2012, edn.), pp. 4-15

[93] Arnold, A.: 'Decoding Space and Time in the Brain', 2011

[94] Paul, J.M.: 'Computational Dreaming', 2010

[95] Tomlinson, B., Baumer, E., Yau, M.L., Alpine, P.M., Canales, L., Correa, A., Hornick, B., and Sharma, A.: ‘Dreaming of adaptive interface agents’: ‘Book Dreaming of adaptive interface agents’ (ACM, 2007, edn.), pp.
[96] Boyer, M., Skadron, K., Che, S., and Jayasena, N.: ‘Load balancing in a changing world: dealing with heterogeneity and performance variability’. Proc. Proceedings of the ACM International Conference on Computing Frontiers, Ischia, Italy2013 pp. Pages
[97] Siegel, J.M.: ‘Why we sleep’, Scientific American, 2003, 289, (5), pp. 92-97
[98] Paul, D.B., and Baker, J.M.: ‘The design for the Wall Street Journal-based CSR corpus’: ‘Book The design for the Wall Street Journal-based CSR corpus’ (Association for Computational Linguistics, 1992, edn.), pp. 357-362
[99] Yu, S., and Chang, E.: ‘Studies in massively speaker-specific speech recognition’: ‘Book Studies in massively speaker-specific speech recognition’ (2004, edn.), pp. I-825-828 vol.821
[100] Xuedong, H., and Lee, K.F.: ‘On speaker-independent, speaker-dependent, and speaker-adaptive speech recognition’, Speech and Audio Processing, IEEE Transactions on, 1993, 1, (2), pp. 150-157
[101] Bobrek, A., Pieper, J.J., Nelson, J.E., Paul, J.M., and Thomas, D.E.: ‘Modeling shared resource contention using a hybrid simulation/analytical approach’: ‘Book Modeling shared resource contention using a hybrid simulation/analytical approach’ (IEEE, 2004, edn.), pp. 1144-1149
[102] Bobrek, A., Paul, J.M., and Thomas, D.E.: ‘Shared Resource Access Attributes for High-Level Contention Models’: ‘Book Shared Resource Access Attributes for High-Level Contention Models’ (2007, edn.), pp. 720-725
[103] Bobrek, A., Paul, J.M., and Thomas, D.E.: ‘Stochastic Contention Level Simulation for Single-Chip Heterogeneous Multiprocessors’, Computers, IEEE Transactions on, 2010, 59, (10), pp. 1402-1418
[104] Suontausta, J., Hakkinen, J., and Viikki, O.: ‘Fast decoding in large vocabulary name dialing’: ‘Book Fast decoding in large vocabulary name dialing’ (2000, edn.), pp. 1535-1538 vol.1533
[105] Biau, D.J.: ‘In Brief: Standard Deviation and Standard Error’, Clinical Orthopaedics and Related Research®, 2011, 469, (9), pp. 2661-2664
[106] Altman, D.G., and Bland, J.M.: ‘Statistics notes: standard deviations and standard errors’, BMJ: British Medical Journal, 2005, 331, (7521), pp. 903
[107] Dugdale, S.: ‘Whats your speech rate?’, 2012
[108] Meyer, B.H., Pieper, J.J., Paul, J.M., Nelson, J.E., Pieper, S.M., and Rowe, A.G.: ‘Power-performance simulation and design strategies for single-chip heterogeneous multiprocessors’, Computers, IEEE Transactions on, 2005, 54, (6), pp. 684-697