Personalized Computer Architecture as Contextual Partitioning for Speech Recognition

Christopher Grant Kent

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

JoAnn M. Paul
Paul E. Plassmann
William T. Baumann

11/20/2009
Blacksburg, Virginia

Keywords: Parallel Architectures, Parallel Processing, Mobile Computing, Language Processing, Brain Modeling

Personalized Computer Architecture as Contextual Partitioning for Speech Recognition

Chris Kent

ABSTRACT

Computing is entering an era of hundreds to thousands of processing elements per chip, yet no known parallelism form scales to that degree. To address this problem, we investigate the foundation of a computer architecture where processing elements and memory are contextually partitioned based upon facets of a user's life. Such Contextual Partitioning (CP), the situational handling of inputs, employs a method for allocating resources, novel from approaches used in today's architectures. Instead of focusing components on mutually exclusive parts of a task, as in Thread Level Parallelism, CP assigns different physical components to different versions of the same task, defining versions by contextual distinctions in device usage. Thus, application data is processed differently based on the situation of the user. Further, partitions may be user specific, leading to personalized architectures. Our focus is mobile devices, which are, or can be, personalized to one owner. Our investigation is centered on leveraging CP for accurate and real-time speech recognition on mobile devices, scalable to large vocabularies, a highly desired application for future user interfaces. By contextually partitioning a vocabulary, training partitions as separate acoustic models with SPHINX, we demonstrate a maximum error reduction of 61% compared to a unified approach. CP also allows for systems robust to changes in vocabulary, requiring up to 97% less training when updating old vocabulary entries with new words, and incurring fewer errors from the replacement. Finally, CP has the potential to scale nearly linearly with increasing core counts, offering architectures effective with future processor designs.

Acknowledgements

Completion of this Thesis is a testament to support I was fortunate to have from many individuals. I would like to thank my advisor, Dr. JoAnn Paul, for her unwavering assistance in guiding the structure of this research, and her encouragement to persevere through frustrations. I also would like to thank Dr. Paul Plassmann and Dr. William Baumann for serving as members of my advisory committee. I also wish to thank my family, both immediate and extended, for a lifetime of love and support. Additionally, I'm especially grateful to my parents, Mr. Dan Kent and Ms. Debbie Kent, and my grandparents, Dr. Eric Force and Ms. Jane Force, for their assistance in making my undergraduate and graduate dreams possible. Finally, I wish to thank Ms. Sarah Lyon-Hill for her support, both near in Blacksburg, and far in Niger.

Table of Contents

Abstract ii
Acknowledgement iii
Table of Contents iv
List of Figures vi
List of Tables vii
List of Abbreviations vii
1 Introduction 1
1.1 Why Contexts 2
1.2 Example Implementation 3
2 Background 7
2.1 Computer Architecture Background 7
2.1.1 The Parallelism Problem 7
2.1.2 Traditional Architecture Classifications 9
2.1.3 Classification of Contextual Partitioning 10
2.2 Language and Speech Processing Background 11
2.2.1 Statistical and Neural Decoders 12
2.2.2 Hidden Markov Models 13
2.2.3 Grammar Model 15
2.2.4 Corpus Selection 16
2.2.5 Why Contextual Partitioning 17
2.3 Brain Modeling Background 18
2.4 Mobile Computing 20
3 Related Work 22
3.1 Architecture Related Work 22
3.1.1 Chip Multiprocessing 22
3.1.2 System-on-Chip 24
3.1.3 FPGAs 26
3.2 Speech Recognition Related Work 28

3.2.1 Dynamic Grammar Speech Recognition 28
3.2.2 Resource Constrained Speech Recognition 30
3.2.3 Multi-Modal Speech Recognition 31
3.2.4 Contextually Partitioned Speech Recognition 31
4 Problem Statement 34
4.1 Vocabulary Definition 35
4.2 Contextual Partition Definition 36
4.3 Experimental Goals 39
5 Experimental Setup 41
5.1 Speech Recognition Suite Selection 41
5.2 SPHINX Environment Creation 42
5.3 Performance Measurement 43
5.4 Coarse-Grained Partitioning 46
5.5 Fine-Grained Partitioning 47
5.5.1 Word Model Configuration 47
5.5.2 Fine-Grained Partition Configuration 48
5.6 Dynamic Corpus Comparison 50
6 Results 52
6.1 Coarse-Grained Partitioning 52
6.2 Fine-Grained Partitioning 54
6.2.1 MLB Fine-Grained Partitioning 54
6.2.2 MLS Fine-Grained Partitioning 56
6.2.3 MLB/MLS Comparison 57
6.3 Dynamic Corpus Comparison 60
6.4 Summary 61
7 Scalability 66
8 Conclusion and Future Work 71
References 73
Appendix A: Vocabularies 79
Appendix B: Annotated List of Figures 85

List of Figures

Figure 1: Contextual Partitioning 1
Figure 2: Task/Thread Level Parallelism 3
Figure 3: Coarse-Grained CP 5
Figure 4: Fine-Grained CP 6
Figure 5: CP with Multiple Outputs 11
Figure 6: HMM Topologies from [32]. 14
Figure 7: TLP Speedup from [38] 17
Figure 8: Brain areas for speech perception from [45]. 19
Figure 9: Realtor/Baseball Fan CP 34
Figure 10: Realtor/Baseball Fan Coarse-Grained CP 36
Figure 11: Realtor/Baseball Fan Fine-Grained CP, Hybrid Models 37
Figure 12: Realtor/Baseball Fan Coarse-Grained CP with GP 38
Figure 13: Realtor/Baseball Fan Fine-Grained CP with GP 39
Figure 14: Decoding Result Example 44
Figure 15: Error/Samples Coarse-Grained Partitioning vs. Unified 52
Figure 16: Error/Training Coarse-Grained Partitioning vs. Unified 53
Figure 17: Error/Samples MLB Fine-Grained vs. Coarse-Grained 55
Figure 18: Error/Training MLB Fine-Grained vs. Coarse-Grained 55
Figure 19: Error/Samples MLS Fine-Grained vs. Coarse-Grained 56
Figure 20: Error/Training MLS Fine-Grained Vs. Coarse-Grained 57
Figure 21: MLB/MLS Sub-Contextual Breakdown 57
Figure 22: Error/Samples Hybrid CP vs. Unified 63
Figure 23: Error/Training Hybrid CP vs. Unified 64
Figure 24: TLP Speedup for SR from [38]. 66
Figure 25: Ideal CLP Speedup vs. TLP Speedup from [38]. 67
Figure 26: Word Acoustic Model Sizes 68
Figure 27: Tri-phone Acoustic Model Sizes 69

List of Tables

Table 1: Online Mobile Applications 21
Table 2: Coarse-Grained Partitions 46
Table 3: Fine-Grained Partitions 49
Table 4: Fine-Grained Partitions 54
Table 5: MLB Sub-Context Training/Sample Contributions 58
Table 6: MLS Sub-Context Training/Sample Contributions with wArea 58
Table 7: MLS Sub-Context Training/Sample Contributions with pArea 59
Table 8: Dynamic Corpus Experiment Results 60
Table 9: Corpus List 79

List of Abbreviations

ANN – Artificial Neural Network
ASMP – Asymmetric Multi-Processor
CAA – Context Activation Algorithm
CASIS – Context-Aware Speech Interface System
CLP – Contextual Level Parallelism
CMP – Chip Multi-Processor
CMU – Carnegie Mellon University
CP – Contextual Partitioning
DLP – Data Level Parallelism
ET – Error/Training
fMRI – Functional Magnetic Resonance Imaging
FPGA – Field-Programmable Gate Array
GP – General Partition
HMM – Hidden Markov Model
HTK – Hidden Markov Model Toolkit
ILP – Instruction Level Parallelism
IP – Integrated Peripheral

ISA – Instruction Set Architecture

MFC – Mel Frequency-Warped Cepstrum

MFCC – Mel Frequency-Warped Cepstrum Coefficient

MIMD – Multiple Instruction Multiple Data

MISD – Multiple Instruction Single Data

MIT – Massachusetts Institute of Technology

MLB – Major League Baseball

MLS – Multiple Listing Service

MPSoC – Multi-Processor System-on-Chip

NoC – Network on Chip

PDA – Personal Digital Assistant

PE – Processing Element

RFID – Radio Frequency Identification

RNN – Recurrent Neural Network

SIMD – Single Instruction Multiple Data

SISD – Single Instruction Single Data

SLM – Statistical Language Modeling

SMP – Symmetric Multi-Processor

SoC – System-on-Chip

SUMA – Single-User Multi-Application

TDM – Time Division Multiplexing

TDNN – Time-Delay Neural Network

TIMIT – Texas Instruments / Massachusetts Institute of Technology (speech corpus)

TLP – Thread Level Parallelism

U-A – User-Application (taxonomy)

VICO – Virtual Interactive Co-Driver

WIP – Word Insertion Probability

WSJ – Wall Street Journal (speech corpus)

1 Introduction

Computer architecture performance has been, and will continue to be, tied to parallelism. We are entering an era of hundreds to thousands of processing elements on a chip [1], yet we have no known parallelism form capable of scaling to that degree. Computer architects agree that new degrees of parallelism are necessary to continue to provide the performance gains expected of computing devices, but they disagree about how this will be achieved [2]. Traditional forms of parallelism, which focus components on mutually exclusive parts of a single unified problem, are difficult to apply to personal computing applications, which tend to be dominated by control flow. This presents a problem, given the landscape of future chip designs. To address this, we investigated the foundation of a computer architecture where processing elements and memory are contextually partitioned based upon facets of a user's life, ushering in a new parallelism method – Contextual Level Parallelism (CLP). Contextual partitioning (CP), the situational handling of inputs, allocates resources differently from current approaches. In CP, the same functionality may execute on many different processors, but the programming can be made simpler because the data can be constrained to mutually exclusive, focused situations. Our overall vision is depicted in Figure 1.

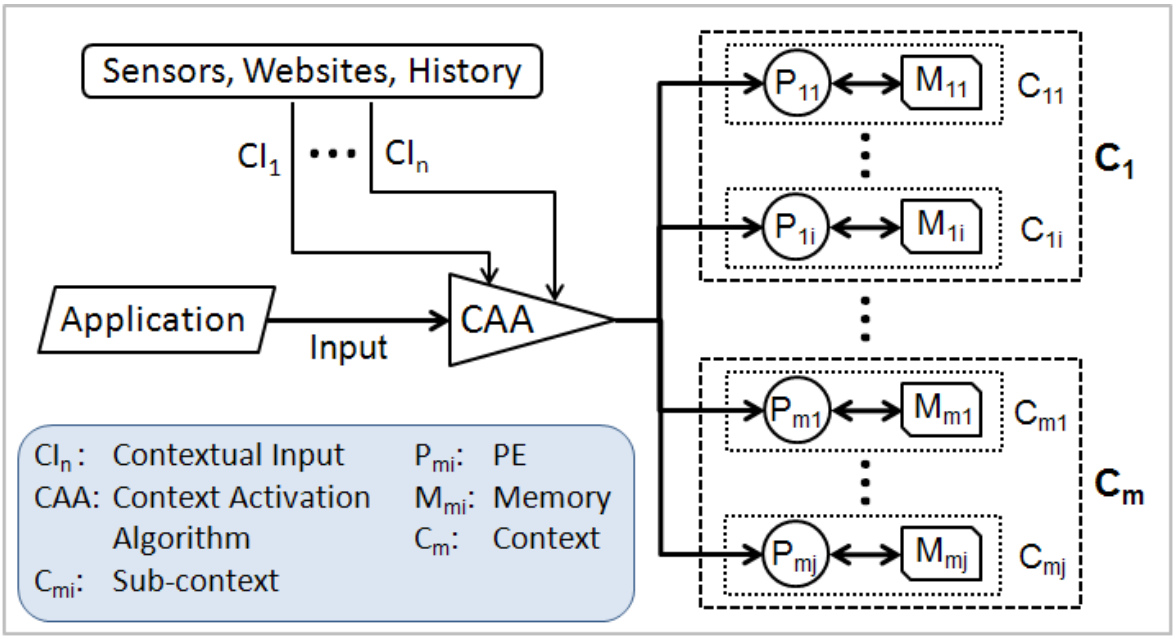

Figure 1: Contextual Partitioning

Figure 1 shows processing elements (P_{mi}) and private memory (M_{mi}) pairings contained within contextual partitions. Resources are first partitioned coarsely by M contexts (C_m), each of which contains finer-grained partitions in the form of sub-contexts (C_{mi}). Thus, the architecture is divided broadly, for example by occupation and hobbies, then more finely, for example by names and places specific to each broad division. Each sub-context is assigned a distinct processor/memory pair for independent computation. Input from an application is passed through a context activation algorithm (CAA), which forwards the input to partition(s) the CAA determines to be appropriate, given the user's current context. The user's context is calculated by the CAA using contextual inputs. Contextual inputs could be data from environmental sensors used to determine where the user is, or what website the user is currently viewing, or what the user historically does given the current time and date. These contextual inputs provide the hard data from which the CAA decides where to forward the application input. Thus, data from an application is processed differently, and by different components, based on context.

1.1 Why Contexts

Contextual processing is inspired by how the human brain analyzes stimuli, where it provides an important role in decision making. Sources such as [3] state that "categorization may be what makes possible human perception, memory, communication, and thought as we know it". Similarly, CP is a form of parallelism that employs simplified, redundant functionality, which is consistent with the massive parallelism found in the brain, as well as its ability to compartmentalize and respond quickly, in real time. Assigning processing elements (PEs) to contexts can result in multiple PEs performing similar tasks under different circumstances. Similarly, components of our brains are filled with overlap [4], but remain distinct by maintaining differences from one another [5]. These components work together, often similarly across multiple tasks, but never identically. CP follows the same concept. Although the partitions might process the same data at different times, the context will be different, as will the processing approach and/or the final output. This defines a new level of parallelism in computer architectures – contextual level parallelism. CLP's approach towards assigning resources is in contrast with Thread or Task Level Parallelism (TLP), where traditional research in parallelism for computer architecture has been largely spent.

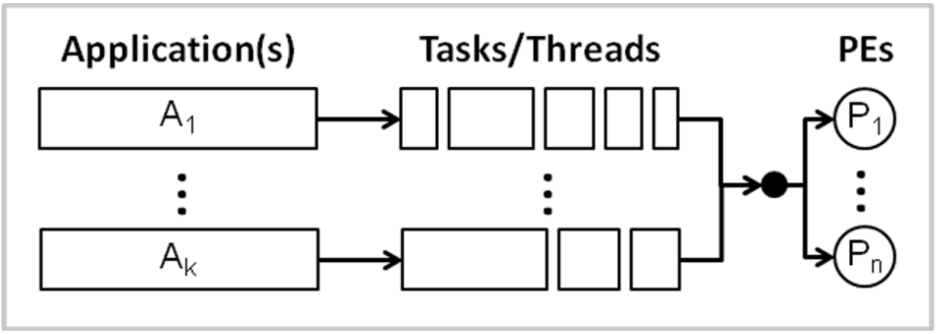

Figure 2: Task/Thread Level Parallelism

TLP's emphasis on dividing tasks into mutually exclusive parts, for parallel processing across multiple PEs, provided a gateway into the multi-core revolution. As the number of PEs on a chip increases however, TLP is limited in its ability to scale. Some architects suggest that "developers should start thinking about tens, hundreds, and thousands of cores now" [1]. At the same time, they also acknowledge that "except for 'embarrassingly parallel' applications, no one really knows how to exploit lots of cheap threads" [6] and that "obtaining scalability for real applications across factors of more than five in processor count is a *major* challenge" [2]. Our architecture proposes an answer, inspired by and consistent with observations about the organization of the human brain. We define three primary questions to guide our exploration:

1) For what applications is a contextually partitioned architecture viable and beneficial?
2) What performance metric(s) should be used to evaluate this architecture?
3) How should the partitioning into contexts be accomplished?

We address these questions through implementation of an application-level example.

1.2 Example Implementation

Our research considers CP for mobile device architectures, leveraging device/owner personalization as a means to define partitions. Mobile devices, such as cell phones or Personal Data Assistants (PDAs), which are tied predominantly to one user, lend themselves well to such personalization. Furthermore, these devices are becoming more ubiquitous, and are covering a wider application base [7]. Our architecture will leverage contexts in order to partition and simplify an application. Instead of adding overhead to communicate between tasks, we add

overhead by repeating the same task in the form of different versions, operating on different data. In order to accomplish this, we need to establish a target application. We focus on applying CP to facilitate speech recognition on a mobile device.

Language processing stands out as a good initial application for investigation for three main reasons. First, language usage is highly situational, and determinant on contextual factors including location, education, occupation, age, and many other variables [8]. This affords us a variety of methods for defining partitions. Second, speech recognition algorithms are search space driven, and their performance is tied closely to how efficiently they can span the vocabulary database while returning as accurate a prediction as possible. The brain is thought to use context to constrain retrieval options in human memory [9]; similarly, our architecture will partition the search space for speech recognition based on context, facilitating faster and more accurate recognition. Third, speech driven user-interfaces for devices, such as cell phones, are desired [10], yet vocabulary complexity is hindered by processing and battery limitations inherent on a mobile platform, as well as the desire to limit user required training of the system.

Our evaluation of the performance of our architecture is defined by the challenges of implementing speech recognition on mobile devices. Speed of recognition represents one metric, as real-time recognition is usually a must, and recognition time is proportional to vocabulary size [11]. We will show that CP allows for scalable speedup in decoding time across increasing PE/context pairings. While speech recognition technology has been focused on speaker-independent systems that require no user training [12], we consider instead a speaker-dependent solution to provide optimal personalization and accuracy. A second performance metric, therefore, must center on minimizing error rate per training required. We will show that CP is more efficient with training, achieving higher accuracy than a unified approach. Finally, personalized vocabularies emphasize the possibility that the system's vocabulary will change over time, as applications or user interests themselves change. Exploring how our architecture handles such change, versus a unified approach, represents a third metric by which we measure performance. We will show that by separating dynamic aspects of a vocabulary, using CP, we are able to achieve greater robustness to changing vocabularies, in terms of retraining time and change in accuracy.

Since partitions are dependent on device personalization, we tailor our research to an example user. We define our user to be a Northern Virginia based realtor who is an avid fan of the Major

League Baseball (MLB) team the Washington Nationals. For our user, two divisions are apparent at first: distinction between work and hobby.

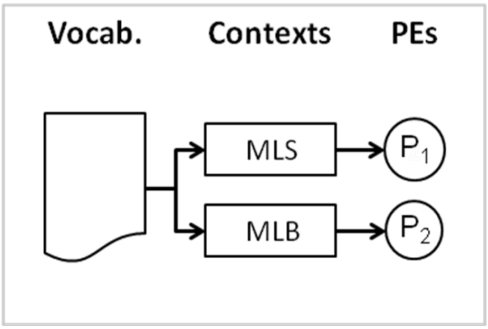

Figure 3: Coarse-Grained CP

Consider a speech interface for a web-browser. Our person would use the internet for both contexts (occupation and hobby), but in different ways. As a realtor he/she might browse the Multiple Listing Service (MLS), a home database, while as a MLB fan he/she might browse the MLB website for team and player stats. These are interesting contexts for several reasons. Real-estate, including the MLS, is a driving market of the economy – interesting many. It also contains a fairly static set of variables (number of bedrooms, bathrooms, regions of the particular MLS's coverage) onto which a dynamic set of listings is mapped. MLB meanwhile, as a sport, garners popularity and represents a particularly dynamic context. Statistics for players and teams change frequently, and even player rosters are modified with regularity as pitchers or other players are rotated either by design, injury or trade. MLB similarly has static variables, such as teams and the cities those teams are located in. This static/dynamic split between our contexts provides a basis for their investigation since people themselves have portions of their life that are both formulaic and random. Designing a system capable of handling both situations is valuable. Furthermore, they represent popular subjects, and thus are immediately applicable to many.

Thus our speech interface for controlling the web-browser could have its vocabulary split between these two contexts, MLS and MLB. Yet there are finer contexts, or sub-contexts, present in both MLS and MLB. For instance, MLS could be subdivided into regions of Northern Virginia and street names, and MLB into team names and player names. We define these two approaches as coarse-grained and fine-grained CP.

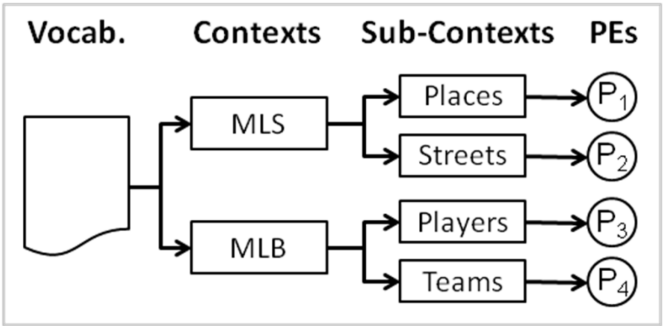

Figure 4: Fine-Grained CP

In conducting this research we will meet the following objectives:

1) Explore CP, both coarse-grained and fine-grained, within Speech Recognition to:
    a. Improve the Error/Training (ET) Curve
    b. Provide robustness for dynamic vocabularies
    c. Leverage greater numbers of PEs, maximizing decoding speedup
2) Define challenges for future work in CP

We begin by discussing background and related work in sections 2 and 3, before progressing into our problem statement, experimental setup and results in sections 4, 5, and 6. We close with a final discussion on scalability in section 7, and our conclusions in section 8.

2 Background

Our research pulls together several disparate topics. First, our goal is to provide the foundation for a new form of parallelism in computer architecture. Second, our focus application is speech recognition, for which our new architecture impacts performance. Third, brain modeling serves as an inspiration for the overall organization and provides insight for next steps beyond the foundation supplied by this work. Finally we explore the expanding mobile computing market, our target area for this research.

2.1 Computer Architecture Background

Our background on computer architecture will first consider parallelism levels driving the market, defining the need for novel architectures such as our own. We then cover classifications of architectures in a traditional sense, before entering a discussion of how contextual partitioning maps onto existing taxonomies.

2.1.1 The Parallelism Problem

Parallelism has been a driving force in computer architectures for nearly as long as the field has existed. Through the years, its forms have evolved from Instruction and Data Level Parallelisms (ILP/DLP), so dominant in the 1980s through the 1990s, into the Thread Level Parallelism (TLP) prevalent with today's multi-core processors [2]. This evolution was necessitated by mounting limitations of ILP and DLP, and the power wall uni-processors were unable to overcome [2, 13]. TLP was championed as the method to utilize the increasing transistor budget, dictated by Moore's Law, to consistently produce processors with greater speedup. TLP functions on division of a task across multiple cores however, and it quickly became apparent that such division is non-trivial for new and old applications alike [13, 14]. Nevertheless, with increasing transistor budgets and consumer demand, researchers and architects continue to devise ways to better exploit TLP in the name of performance. Amdahl's Law shows us that even minute amounts of sequential code can have drastic implications on the

performance of software on parallel architectures [2, 13]. Furthermore, this technique has many variables, such as underlying hardware, parallelization strategy, and compiler choice, all of which affect the performance of the final product. These challenges culminate in the result that each decimal order of magnitude increase in parallelism requires intensive redesign of parallel code [14]. Scalability also becomes a concern with larger core counts. As Hennessy and Patterson point out [2], "multiprocessors that scale to larger processor counts require substantially more investment". Nevertheless, core counts of processors are projected to continue increasing, leaving researchers with an important, but all too familiar question: how to best generate and leverage parallelism with an increasing transistor budget, and in turn, an increasing number of cores.

We propose an architecture leveraging a different form of parallelism, based upon the knowledge that the ways in which we use a given device are dependent upon the context of the user. This presents the possibility of allocating cores to focus on the different aspects of a user's life and personality. We are exploiting task-level redundancy in order to overcome the sequential barrier, predicted by Amdahl, as more complex algorithms encounter sequential fractions. This will divide the application space across cores based on the situation of the user, resulting in redundant, specialized, simplification. Information such as whether the user is at home or at work, and what their occupation and hobbies are, provide context as to the ways in which they might use a computing device. By leveraging this contextual information, for applications based on a central database, it becomes possible to divide the database for context-dependant access. Instead of focusing solely on dividing computation effort across multiple cores, this technique suggests that context can be used to change *how* the data is processed while limiting the search space of an algorithm. This approach falls under the class of context-aware architectures, a recent computing style that has captured a great deal of attention, mainly in the forms of intelligent rooms [15, 16] and smart personal devices [17-20]. Our vision, encapsulated in Figure 1, suggests the possibility that input from an application may be forwarded to multiple partitions, and thus processed different ways simultaneously, if the CAA determines multiple contexts are, or may be, active. This deviates from most traditional computer architecture work, leading us to consider how one might classify a contextually-partitioned approach.

2.1.2 Traditional Architecture Classifications

A question to consider is how our contextually partitioned architecture would compare with existing classifications. Traditionally, computer architectures have been categorized by Flynn's taxonomy based on their implementation and handling of instruction and data streams – specifically the parallelism therein. This taxonomy breaks architectures into four groups: single-instruction single-data (SISD), single-instruction multiple-data (SIMD), multiple-instruction single-data (MISD), and multiple-instruction multiple-data (MIMD). SISD architectures represent traditional uni-processor systems that dominated computing up until the recent multi-core trend [21], where MIMD processors took hold. SIMD processors have consistently held a niche market consisting mainly of vector and array processors for very specific applications, graphics processing for instance. MISD meanwhile is not as well defined, and has garnered little attention. Other researchers have proposed further division of Flynn's taxonomy. While Flynn makes an important breakdown of architectures based on structure, the levels of computing between SISD and MIMD are numerous, and more fine-grained classification could offer better direction for future research, and further levels of parallelism to explore. Some researchers propose classifying architectures based on the number of intended users, and the number of intended applications [22]. This introduces the User-Application (U-A) taxonomy, which defines single/multiple user (SU/MU) and single/multiple application (SA/MA) for a total of four classifications on top of those defined by Flynn. For example, most processors in today's personal computing would fall under the SUMA-MIMD class, where the computer is used by one person, the owner, for multiple tasks, leveraging multiple cores. Furthermore, the U-A taxonomy could help to define the missing portion of Flynn's taxonomy – MISD architectures.

MISD architectures apply multiple heterogeneous types of instructions to the same piece of data, implying that the system either produces multiple outputs for the same data, or that some unification algorithm is present to choose from among the produced outputs. Some argue that pipelining is an example of MISD [23], with different parts of an instruction representing multiple instructions, and the data passing through each pipeline stage representing the single data. Flynn's own definition of taxonomy, however, places pipelining clearly under SISD [21], and given that pipeline stages can modify the data presented, it is questionable whether one could claim that each stage is processing the same data. The best examples of MISD architectures are specially built systems that perform pattern matching, or similar tasks, where a single data must

be tested against a large number of classes [24, 25]. Such systems might test the same data against multiple patterns simultaneously throwing out results until a match is found.

We'll now consider how a contextually partitioned architecture might fit within both Flynn's taxonomy as well as the U-A taxonomy. While we find we are able to classify CP with the U-A taxonomy fairly well, Flynn's taxonomy is more elusive, but our goal is to create a MISD-style of computing, which we believe is motivated by brain inspired computer organization.

2.1.3 Classification of Contextual Partitioning

This research focuses on the domain of SUMA computing, defined by the U-A taxonomy, due to its prevalence among the mobile market. Devices such as cell phones and PDAs are examples of computing technologies that are highly personalized to the user, yet the architectures of these devices remain incredibly uniform among their user base. Our research of allocating processing cores and resources to specific subsets of a user's personality will address this inconsistency – but how does it fit within Flynn's taxonomy? Such an approach should not be seen as detracting from the gains made through TLP, but rather as an extension. Although we only consider single-processor/single-context pairings, one could envision our system applying multiple cores per context to take advantage of both CLP and TLP. In such a situation, contexts themselves would be defined as SISD or MIMD. The system as a whole, however, could have either a single partition processing the input data, or multiple. Should a single context be active, the system as a whole would be defined by the structure of its active partition. However, should multiple partitions process simultaneously, we instead have a MISD computing architecture. CP presents the possibility that multiple, differing, matches may be returned for the same input.

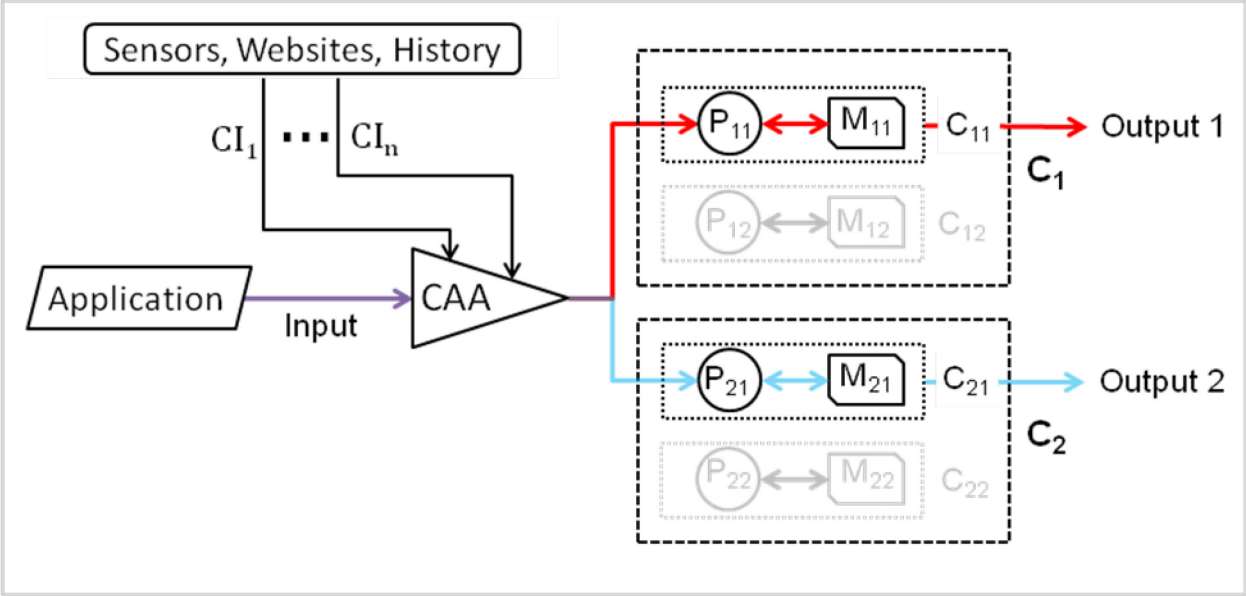

Figure 5: CP with Multiple Outputs

Figure 5 depicts a multiple output situation. Here, data is passed into the CAA, which given the contextual inputs, determines that two sub-contexts may apply in the user's present situation. Thus, the data is forwarded to both sub-contexts for simultaneous processing under the separate sub-contexts, resulting in two distinct outputs. Although this research does not approach the case of multiple outputs, we believe it will be a defining challenge worthy of further work, and our research will bring us to the brink of such an architecture. For our purposes, it is sufficient to examine the architecture's viability while stipulating that only one context be active at a time, leaving a context selection algorithm and output handling for the future. With that in mind, we now direct our attention towards an overview of speech recognition.

2.2 Language and Speech Processing Background

Speech processing has been a premier research area in computing since the 1950s [26]. It became apparent, however, that this was a task that while trivial for most humans, is difficult to break down algorithmically for computers with present architectures. For our purposes, we are considering speech recognition purely from an implementation point of view, and not an algorithmic one. By this we mean that we approach speech recognition as we anticipate an end user would, rather than from the perspective of an expert in the field. Nevertheless, it is helpful

to have a basic understanding of the many variables, ranging from environment to number of speakers to decoder type, associated with designing a speech recognition system. Optimizing these variables is a challenge that many have approached, leading to a multitude of groups proposing different approaches to the many facets of the problem.

2.2.1 Statistical and Neural Decoders

Two principle classes of decoders emerged from early work in speech recognition: statistical and neural [27], both touting their own flagship solution. For statistical this was the Hidden Markov Model (HMM), while neural championed the Artificial Neural Network (ANN). Both techniques center on turning recognition into a pattern matching problem, solvable by computers, and each having its own strengths and weaknesses.

HMMs are well suited for handling variability in speech signals [28], particularly signal duration, since they operate on statistical properties of the signal as a whole. This is important because people rarely speak words at the same speed consistently; without this feature, errors would be incurred simply from saying a word slightly faster or slower. Unfortunately, HMMs also function by making generalizations in the recognition process. First, they assume output independence, meaning that each output is considered to be unrelated to previous outputs. Furthermore, each output's computation is broken into states, and each state is assumed to be independent from previous states to save computation time. Both of these are unrealistic, since language is often related to what has been said in the past. HMMs also assume acoustical vectors follow a Gaussian distribution, which may not always be the case, and have difficulty dealing with noise [29, 30]. Some enhancements to HMMs have been made to better address this, and, nonetheless, HMMs have become a gold standard for speech recognition. Many of the most prominent decoders are based upon HMMs [30].

ANNs meanwhile are capable of approximating any continuous function with a simple structure (not just Gaussian) and require none of the strong assumptions HMMs do. They also are more capable when dealing with noise. Unfortunately they are not as well equipped to handle variability in the input signal duration. Networks such as Time-delay Neural Networks (TDNNs) and Recurrent Neural Networks (RNNs) have been used to try and answer this problem, but they are not perfect, and for this reason, ANNs are often found paired with HMMs when used for recognition [28, 30]. These hybrid architectures can perform well, but tend to have a limited

vocabulary because training becomes difficult and computationally prohibitive [28]. As such, despite the advantages of ANNs and hybrid systems, HMMs have remained as the premier solution; "a number of people have warned that HMMs would hit a plateau more than ten years ago, but that has not yet come to pass" [31]. Our research utilizes the SPHINX decoder from Carnegie Mellon University (CMU), which is based upon HMMs, and thus we end our discussion on ANNs here.

2.2.2 Hidden Markov Models

Markov Models define a structure to imitate some stimuli – in this case speech. The model is defined by N states, while a state transition probability network defines the likelihood of the given model progressing from state 'A' to state 'B' [29]. The topology of the Markov Model defines the characteristics of the state transition network. In some cases all the states are interconnected to each other (an ergodic model), in other situations some may not be reachable from others (a non-ergodic model) [29, 32]. In addition, the template includes a set of M observations. These observations are the dividing point between two categories of Markov Models. An observable system is one in which the states of the template correspond to physical, observable, events – the M observations. A hidden model, our HMM, is therefore one whose states represent some unknown situation, and the observations are probabilistic functions of current state [29].

When used for speech recognition, HMMs are used to model some defined unit of speech; a different model could be employed for each word in a vocabulary, phonetic units of words, or various other possibilities. No matter what unit is chosen, each HMM must be trained with a number of samples to be an effective model. For smaller vocabularies, whole word HMMs are not unusual, but as the vocabulary increases, the cost in training for such large models becomes unwieldy, and differentiation between similar words becomes more difficult. For this reason, larger vocabularies tend to imply models with smaller units of speech [29, 32]. Phonemes, the basic building blocks of speech, are commonly used because they limit the number of models to about forty. Larger vocabularies can be used with phoneme HMMs since a phoneme trained with one sample of speech can be used to decode that same phoneme with a different sample of speech. Unfortunately, the small set of phonemes falls prey to difficulties in differentiating similar words when dealing with large vocabularies. For this reason, the most oft-used unit for

large vocabulary recognition is the tri-phone, or context-dependent phoneme [32]. These are simply phonemes that are defined with reference to the previous and next phoneme in the transcription. This gives the decoding process some reference to work with, allowing for even greater vocabulary range and diversity, but does come at the cost of increased training. Enumerating all possible tri-phones with a normal set of phonemes creates a list well over one-hundred thousand. To compensate, an extra step in the training process is often used to tie similar tri-phone states together into what are called senones. The number of senones is another compromise between distinctiveness and trainability [32].

As mentioned earlier, the topology of an HMM defines the state transition network. This too becomes a compromise; more connections for each state implies more training and greater decoding time. Fortunately, speech is a chronological signal, and this can be used to limit the transition network. Transitions, for instance, that return to a previous state, are not needed, thus ergodic models are not normally used in this application. Instead, three other topologies are commonly employed: linear, left-to-right, and particularly Bakis [32], depicted in Figure 6.

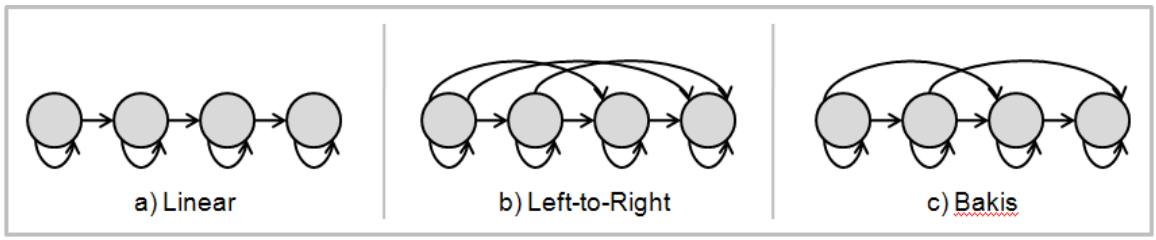

Figure 6: HMM Topologies from [32].
Illustrated under "Fair Use" copyright guidelines.

The linear topology allows each state only to either return to itself or progress to the state immediately following it. This is the simplest of the three, and thus the easiest to train and fastest to decode, but is inflexible in that it has difficulty handling compressions of portions of a segment of speech due to a quick utterance or slang pronunciation. At the other extreme is left-to-right, which allows each state to return to itself, or to any state after it, rather than just its immediate neighbor. Flexibility here comes at the cost of computational and training effort. The most commonly used topology, Bakis, falls between these two, where each state can return to itself, progress to the next state, or skip the next state instead progressing to the following state. This has been found to be a successful compromise, both allowing for flexibility in pronunciation and limiting costs associated with the topology [32]. These are just a few of the

many variables associated with model generation. The end product however, is a firm mathematical method of classification.

2.2.3 Grammar Model

Speech decoding is primarily composed of two main steps. First an input is preprocessed, usually using mel-frequency ceptrum, and statistical features are extracted. Second, the obtained features, or observations, are compared against trained models to determine the most likely set of states to have produced those observations [27, 29]; the most likely candidate is returned as the hypothesis. Two models are typically used in speech decoding. The acoustical model, created with HMMs through statistical features of training samples, we've already discussed. The grammar model is the second half of this process. As mentioned earlier, HMMs assume output independence. In order to help offset this assumption, and to obtain improved accuracy, a grammar model is employed. The grammar model adjusts the decoding process, such that given a series of words, hypotheses are constrained to words that the system has been trained to expect to follow one another. A simple example of this would be if one was asked to "turn", one might assume it likely that the entire phrase may end up as either "turn left" or "turn right", and similarly it would be unlikely that it would end as "turn piano". Thus, given a word "turn" the grammar model would adjust the decoding process to favor a hypothesis of "left" or "right" rather than "piano". A common form of language model is the N-gram which generates probability for n-word combinations (i.e. a 2-gram with word models would generate probabilities for each combination of 2 words). The weight the grammar model carries in the recognition process versus the confidence scores of the hypotheses themselves is yet another balancing act in obtaining maximal performance.

Grammar models constrain the recognition process, and play a significant role in achieving high system accuracy. With contextual partitioning however, a N-gram grammar model would not be solely sufficient. Particularly with fine-grained CP, there could be significant interplay between different acoustic models, and since grammar models are tied to the recognizer and its associated acoustic model, a higher level grammar system would have to be employed. For this reason, we ignore the grammar model in this research, instead focusing on accuracy and training efficiencies of splitting the acoustic model. Towards that end, we also focus on isolated word recognition, where only one word is recognized at a time. Although continuous speech

recognition, where the user can speak freely to the system, is desirable, it also places a great deal of emphasis on the grammar model, which is not the focus of this research and should instead be considered in later work.

2.2.4 Corpus Selection

Another consideration is what audio corpus to use to train a speech recognition system. Two primary classifications of acoustic models are used: speaker-dependent and speaker-independent. Speaker-dependent corpuses are built from audio samples from the intended user – thus the user trains the acoustic model with his or her own voice. Speaker-independent systems are trained over a variety of speakers with the aim of making a corpus that will work for any user, without that user needing to train the system. Speaker-dependent acoustic models have the benefit of naturally aligning themselves with the user's speaking tendencies, offering superior accuracy, and the freedom for the user to train with only their intended vocabulary [12]. Speaker-independent corpuses are often large (thousands of words), such as the Texas Instruments/Massachusetts Institute of Technology corpus (TIMIT) [33], Hub 4 – a business broadcast news corpus [34], or the Wall Street Journal (WSJ) corpus [35], and/or licensed through the Linguistic Data Consortium. The benefit of not requiring user training is considerable, but it comes at a cost of accuracy and flexibility. Speaker-adaptive speaker-independent attempts to make a compromise, allowing for the use of a speaker-independent corpus that adapts itself to a user's voice through training. Minimizing training is highly desirable for speaker-adaptive and speaker-dependent corpuses alike [36].

For personalized user devices, like we are considering, where it would be desirable to have a vocabulary tied specifically to the applications that user is interested in, speaker-dependent systems would be beneficial if training time could be reduced – even ultimately folded into normal operation of the system by just the kind of parallelism we are exploring: splitting a single input stream into different paths. Also, for some groups, such as the military or medical industry, ultimate accuracy may always supersede training costs. Furthermore, it seems likely that an approach that reduces training time for a speaker-dependent system would also reduce training time to adapt a speaker-independent corpus, should a suitable one be available for a user. Given these considerations, and the limited availability of specialized, public, speaker-independent corpuses, this research will focus on creating our own, speaker-dependent databases.

2.2.5 Why Contextual Partitioning

The primary challenge with speech recognition is that as the vocabulary grows it becomes more and more difficult to achieve high accuracy and real-time decoding. This problem becomes more apparent when considered with mobile device architectures. Larger vocabularies are desired to facilitate more natural human-computer interaction over an increasing application set. Yet larger vocabularies require smaller recognition units, which consume more training, and are more likely to contain confusable vocabulary words, hampering accuracy. Furthermore, decoding time increases linearly with respect to vocabulary size [11], and larger vocabularies tend to place more strain on the grammar, resulting in grammar models that take up more storage space and processing power themselves [37]. While algorithmic improvements, such as in [11], can reduce overall decoding time, the fact remains that smaller vocabularies are able to be processed faster. Harnessing multiple PEs with TLP is one option, yet research shows that decoding speedup does not scale well with large numbers of PEs [38], as seen in Figure 7.

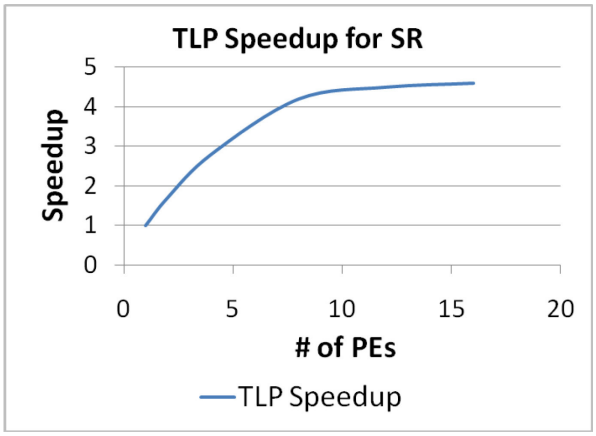

Figure 7: TLP Speedup from [38]
Illustrated under "Fair Use" copyright guidelines.

Given the limitations of TLP, we propose approaching a different level of parallelism based on the way the human brain is believed to process information. Contextual partitioning is thought to play a major role in our brains ability to process stimuli, we aim to merge this concept with computer architecture. To better understand contextual partitioning, we'll now explain its role in the human brain.

2.3 Brain Modeling Background

Research in brain modeling has provided numerous insights to the structure and function of the human brain. Much of this effort has been spent attempting to understand parallelism and overlap in the brain. A pinnacle technology for advancement in this area is functional magnetic resonance imaging (fMRI). fMRI monitors blood oxygen levels in the brain, which can be used to map what components in the brain are active at a given time. This allows for non-invasive studies of brain organization that were previously infeasible [39]. The mapping capabilities of fMRI allowed for research such as [5], where a comparison of components of the brain utilized for sentence formation versus melody formation was performed. Results depicted large amounts of common components between the two tasks, but distinct elements for each in addition.

There is also a notable difference in brain activity in some areas between old and new memories, or those that are assumed to be either old or new correctly or incorrectly; other areas perform identically regardless of memory age [40, 41]. Additionally, some elements in the brain that are activated in memory retrieval are triggered independent of the context of the memory; other areas, such as those associated with visual recognition, activate specifically for memories closely associated with a given context [41]. Similarly, deductive reasoning activates predominantly right-brain areas, while probabilistic reasoning activates mostly left-brain areas [42]. Portions of the brain are also shown to prepare themselves for information if the type of information can be anticipated. This is believed to constrain retrieval options, and lead to faster recognition times [9]. These works provide substantial support for the contextual divisions we propose for our computer architecture, including the foundation of its larger overall vision.

The brain is known to have alternate components develop to further support or even take on the roles of damaged ones in impaired individuals [43, 44], but this purpose is perhaps secondary to the parallel processing provided by this overlap for healthy people:

> To process the information that it needs in order to function, the brain employs many structures simultaneously. In other words, it operates on parallel circuits. The same piece of information may be processed simultaneously in several 'centres' of the brain…The brain's centres must therefore be designed more like strategic crossroads or points through which certain data must pass in order to be processed, rather than like the sole sites where certain functions are executed [4].

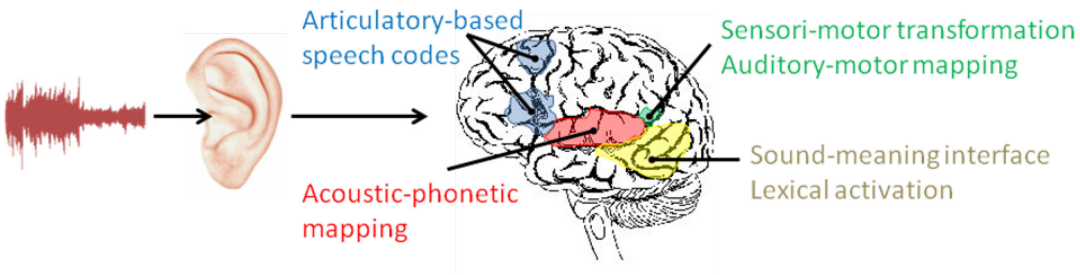

Figure 8: Brain areas for speech perception from [45].
Illustrated under "Fair Use" copyright guidelines.

Ear image used from Public Domain, source: Hardouin, P. *Human Ear.* Feb. 5th 2009 [cited 2009 Oct. 20th]; Available from: http://openclipart.org/media/files/bionet/10903.

Brain image used from Public Domain, modified from: *Free brain-2 Clipart.* Mar. 26th 2008 [cited 2009 Oct. 20th]; Available from: http://www.freeclipartnow.com/science/medicine/anatomy/brain-2.jpg.html.

Figure 8 above shows the many regions activated in the brain for speech recognition. While many of the areas are lateralized to the left hemisphere, the acoustic-phonetic mapping region has bilateral participation. "The fact that imaging studies show bilateral activation does, of coarse, not imply that computations executed in left and right core auditory cortices are identical – there are, presumably, important differences in local computation" [45]. The same authors go on to discuss the emergence of two processing streams in speech decoding – a 'what' and a 'where'/'how' pathway. The evidence for overlap in parallel processing in the brain is clear, yet the precise division of labor, whether it be contextual or not, and the resolution of multiple outputs is largely undefined. "One goal has to be to begin to specify what the computational contribution of each cortical field might be" [45].

Relating the findings in brain modeling to computer architecture, it seems reasonable to organize a computer such that parts could potentially work similarly on the same data to arrive at a more accurate final result(s). Computers have focused on dividing tasks into mutually exclusive parts for parallel processing. Research discussed above suggests the brain does not solely take this approach, rather processing data in different ways based on contexts. While a great deal is unknown about exactly how the brain processes stimuli, the limitations computer architects are facing with TLP we believe justify researching CLP. Thus, we explore the concept of a contextually partitioned architecture, inspired by activity seen in the brain, by leveraging multiple speech recognizers, each with a distinct vocabulary and acoustic model, which will activate during specific contexts. Contextual activation requires personalization of the

architecture however. To meet this personalization requirement we turn our focus towards mobile computing architectures. We will now cover a background in mobile computing, in order to justify our emphasis on the mobile market.

2.4 Mobile Computing

Trends in both consumer purchases and research state that the mobile computing market has expanded drastically, particularly with the increase in internet enabled devices [7, 46]. Furthermore, the applications used on these devices are covering a broader range. Cell phones for instance are now commonly internet enabled, and offer entire application stores featuring products from games to office productivity. Products like Apple's iPhone are becoming less like a phone, and more like a general purpose computing platform. The growth of the mobile market is expected to continue into the foreseeable future, driving research efforts to meet challenges imposed by mobile devices. Applications are fueling the success of these devices, yet the architecture of mobile platforms is constrained in processing power and battery life. Enabling computationally intensive applications on finite mobile resources is a major challenge in the industry.

A promising approach towards implementing complex applications on networked mobile devices is offloading the applications to remote servers. This thin-client approach, also known as cloud computing, removes the need for powerful hardware at the client level, and leaves the computationally intensive tasks up to remote locations that are not constrained as the mobile device is. A wide variety of application sets already have remote online equivalents as demonstrated in Table 1.

Task	Online Application
Office Productivity	Google Docs, Office Live
Gaming	Flash Games, GameTap
Graphics Editing	Adobe Photoshop Express
Video Editing	Adobe Premier Express
Internet Radio	Pandora, Napster Mobile

Table 1: Online Mobile Applications

As these applications become remote and internet-based, the question of what should be done with the processing left at the client remains. Designers and consumers alike are interested in new, more powerful, user interfaces. Cell phones have evolved from the numeric keypads of old, to QWERTY mini-keyboards and styluses, and most recently touch screens. Speech recognition is an enticing interface form, as it is a natural form of communication for people. It also is faster for strings of more than a few keystrokes [47], and lessens dependence on cramped keyboards and other manual input forms. Yet as discussed previously, speech recognition is computationally intensive, prohibitively so for large vocabularies.

Some may argue that speech recognition should too be offloaded to the network. Consider mobile internet however – the bandwidth constantly changes based on location, there may be glitches that occur from time to time, or the connection may be lost all together. These annoyances would detract from the effectiveness of a user interface, and interfere with user acceptance of a device. Similarly, security of embedded devices is a mounting concern [48]; by keeping the user-interface local, it makes it easier to secure that aspect of the device. Given the emergence of mobile computing as a premier computing market, the desire for speech driven interfaces, and the gap between parallelism in the brain and in computer architectures, we believe our research has a reasonable foundation. Before laying out our approach, we'll first consider previous work related to our research.

3 Related Work

This section provides an overview of recent work in computer architecture and speech recognition, since we make contributions in each of these areas.

3.1 Architecture Related Work

Computer architecture development is driven forward by two primary forces. First there is the consumer demand for faster and cheaper electronics, and second there is Moore's law, resulting in ever-increasing transistor budgets. Since uni-processor architectures hit the power wall, architects have been searching for new ways to use the transistor budget to maximize performance for the consumer. We focus on three main areas of the architecture community here. Chip multiprocessing is the art of multiprocessor design. System-on-Chip focuses on designing packages of multiple chips specialized towards a specific application set. Field Programmable Gate Arrays (FPGAs) meanwhile lead the reconfigurable computing community as a method of customizing computing resources at run-time. Each of these areas has limitations when considered for mobile devices however. Our architecture will address their shortcomings.

3.1.1 Chip Multiprocessing

In order to maximize parallelism speedup, architects have considered a number of ways to divide chip real estate into different cores. In the general purpose computing community, most products currently use symmetric multiprocessors (SMPs). SMPs simplify the design step by requiring all cores on the chip to be the same, but lack of specialization in core layout also leads to inefficiency. Asymmetric multiprocessors (ASMPs) have been shown to provide increased performance at reduced power cost, and are touted as being the future of chip multiprocessors (CMPs) [49]. ASMPs leave a number of questions up to the designer however, and obtaining optimal designs are the subject of many research papers.

One approach towards designing ASMPs considers leveraging many small and simple cores to execute the parallel parts of application(s) while just a few larger cores to process sequential

parts. Research in [50] details such an approach, finding that accelerating sequential portions of a program using this method helps to mitigate the effects of serial code noted by Amdahl's Law. Designing the chip to handle both serial and parallel portions of code effectively also leads to greater power efficiency overall, shown in [51]. Others tout ASMP designs that utilize chips of a wider range of processing power, rather than just the pairings of small/parallel and large/sequential. The authors in [52] note that this approach takes advantage of the dynamic computational requirements within threads themselves. Threads can be assigned to cores based on their current processing demand. Threads that would benefit the most from additional processing power would be assigned to the most powerful cores. Should a thread enter a section of code that would run well on a lesser core, it could be migrated to allow access to the high-power core for others.

Both of these approaches rely on the ability to migrate tasks between cores of varying complexity. Migration requires that each core be able to perform the same fundamental set of instructions required by the thread. This means, for the most part, that these systems employ a single Instruction Set Architecture (ISA) amongst the cores. Alternatively, one can configure ASMPs to leverage application specific processors, where multiple ISAs may be leveraged. This requires the designer know ahead of time what tasks the device is expected to perform, and then allows the architect to select processors specifically suited to execute those tasks. This also restricts the flexibility in thread assignment, placing the scheduling burden on the designer to ensure it is performed optimally. Such specialized designs also limit reusability across multiple products or versions of a product, making cost of design a concern. Use of application specific processors with extensible instruction sets [53] allows for use of a common ISA as much as possible, while also enabling extension of the processors with application specific instructions. Reusability is enhanced since the extended instructions can be removed or changed across designs, and performance is enhanced over non-application specific designs through specialization of cores and scheduling.

For workloads that change, and unspecified at the design level, multiple-ISA designs are difficult to implement. Yet, it is similarly difficult to choose how many fixed cores of different granularities to implement as well. While ASMPs provide greater flexibility in this regard than SMPs, an ASMP that is poorly configured for a task will nonetheless be outperformed by one that is better suited. To address this concern, researchers are interested in flexible core

architectures where a CMP is designed with a base set of cores, often symmetric, that are capable of combining with one another to create varying levels higher performance cores [54, 55]. This introduces a number of run-time concerns that must be addressed. The scheduling algorithm must now determine what chip layout would best suit the workload and decide whether the cost of combining cores is worth the delay. Minimizing overhead for core configuration is another goal as well.

While CMP design is focused on how many and what type of cores to provide on a chip, the general trend is increasing core counts. Thus, CMP design is largely operating under the assumption that we will have the parallelism necessary to leverage these additional cores. Scalability of this kind has not been demonstrated by TLP. At the very least, a fundamental change in software design will be necessary to facilitate TLP at high core counts. The alternative is to determine new levels of parallelism. We choose to pursue the latter approach, defining contextual partitioning as a gateway to contextual-level parallelism.

3.1.2 System-on-Chip

For markets outside of general purpose computing, particularly the mobile market, architectures are often specialized to particular tasks. Integration of all the parts necessary for a particular task or set of tasks into a single chip package leads to a System-on-Chip (SoC). Challenges in SoCs differ from those in general purpose computing in three primary ways:

1. Emphasis on real-time computation
2. Fitting computational requirements into limited power and area budget
3. Stringent limitations on cost

While performance is an issue for all computing markets, maximizing performance, as is often the goal of general purpose computing, and real-time performance of a set of tasks are not equivalent constraints. Similarly, while reducing power consumption is always a boon, with batteries entering the picture, squeezing extra performance at the cost of power is a more costly decision for SoC design than general purpose computing. Finally, the mobile and embedded markets SoCs are prominent in are highly cost driven, limiting design decisions and implementable technologies. The performance/power struggle the SoC market faces led to early

adoption of multi-core architectures, particularly asymmetric; yet, challenges in leveraging asymmetric Multi-Processor System-on-Chips (MPSoCs) remain prevalent.

A primary goal for MPSoC design is effective scheduling of tasks for real-time execution and/or minimal power consumption. Various methods have been proposed to tackle this area. Research in [56] describes using a combination of static and dynamic scheduling algorithms to enable real-time performance of parallel multi-media tasks on a MPSoC architecture. Sets of mutually exclusive tasks are mapped to static processor assignments, and a time division multiplexing (TDM) dynamic scheduling algorithm performs time slicing between active tasks sharing a processor. This approach attempts to compromise between the efficiency of an optimal dynamic algorithm and the simplicity of static scheduling, and offers a means to "guarantee a requested minimum throughput and maximum latency, while minimizing the usage of processing resources" [56]. Other research focuses on scheduling policies designed not simply to reduce chip energy, but to minimize hot spots and temperature gradients on the chip, thereby improving device reliability and reduce cooling costs [57]. Still further research considers the deviations in transistor performance from expected behavior [58], using this knowledge to implement statistical probabilities on the effectiveness of different scheduling algorithms.

A primary challenge in scheduling on MPSoCs is that worst-case execution time, an important metric in scheduling algorithms, depends on memory access time. With uni-processors, this can be modeled fairly well, but as the number of PEs increase, contention for fixed memory resources, usually the memory bus, leads to variable access times. Some research focuses on attempting to model memory access variability for use in scheduling algorithms [59, 60], but a deeper problem stems from the knowledge that shared memory bus designs are failing to scale with larger numbers of cores [61]. Network-on-Chip (NoC) is an emerging technology intent on replacing shared bus architectures. NoC applies a computer networking approach towards integrated peripheral (IP) communication in a MPSoC. IPs communicate by sending packets into the routing network, and the packets are forwarded by switches, based on a routing algorithm, to their eventual destination. Typically each IP has its own switch and thus by adding additional IPs, such as additional PEs, the network itself expands to better handle the additional load. Design of NoC architectures has been tackled by many groups [61-63], while others are implementing knowledge of the traffic of a NoC into scheduling algorithms [64] to improve throughput and utilization. Yet NoCs must utilize minimal area and power while providing

enough throughput and low latency to support real-time tasks. Comparisons of different NoC architectures based on these metrics are also important [61].

The greatest challenge with MPSoCs is making design decisions considering all the variables involved (scheduling and operating system, PE choice, memory architecture, etc.). Tools that model these variables effectively for a designer are greatly desired, but are similarly overwhelmed when the possibilities are enumerated. Abstraction becomes necessary, but determining at what level abstraction can be performed without sacrificing model integrity is largely an unsolved problem. System-level abstraction [65-67] is desired by designers, but ultimately constrains one or several of the variables, fixing either PE options, limiting memory architectures, or failing to explore different scheduling algorithms or operating systems. Nevertheless, this is a hurdle that will have to be overcome in some fashion in order to reduce design costs of MPSoCs for future generations of devices. Furthermore, MPSoCs have largely been successful due to their ability to specialize hardware for particular tasks. While this is powerful, many mobile devices are becoming more general purpose. Cell phones are becoming more identifiable as highly compact laptops than as phones. With generalization such as this, specialized hardware becomes more and more difficult to implement. Our architecture proposes a platform that can be personalized to the user, and achieve greater performance on mobile devices through such personalization.

3.1.3 FPGAs

FPGAs offer the designer flexibility both through programmability, and the ability to design specialized hardware and soft microprocessor cores on a single package. The primary benefit of FPGAs comes from their ability to perform parallel portions of a task in specialized hardware, thus generating speedup, while also providing a cheap platform for development and deployment. Many highly parallel computational tasks have been implemented on FPGA fabric including: Fast Fourier Transform [68], calibration algorithms for digital beam forming antennas [69], and plane wave filters for signal processing [70]. The efficiency of FPGAs in tasks like these led to interest in using FPGAs as coprocessors, alongside a main processor. For applications with portions of highly parallel but computationally intensive tasks, offloading work to a specialized FPGA coprocessor can reduce computational time. Implementation of

application-specific FPGA coprocessors can be found in fields such as image processing [71] and Reed/Solomon encoding [72].

Others point out that this so-called loose processor-coprocessor coupling can result in significant communication overhead reducing potential performance, and that implementing a more generic coprocessor support at the processor level is a better approach [73]. This is done by extending opcodes of the processor to support coprocessor interaction. Generic support of coprocessors introduces the possibility of reprogrammable coprocessors, where the FPGA is configured to support different tasks at runtime [74]. Thus, while FPGA programmability is powerful as a design tool for evaluating different product concepts, researchers are also interested in leveraging FPGA reprogramming at run-time. The concept, known as reconfigurable architectures or configurable computing, aims to tailor the hardware of a system based on the current task(s). Yet reconfiguration incurs overhead; thus, limiting this overhead is a concern.

Partial reconfiguration is the technique of reconfiguring only necessary areas of an FPGA, leaving static areas intact and continuing to function during the reconfiguration. The authors in [75] note that partial reconfiguration allows for devices with mutually exclusive tasks to be mapped onto a smaller chip area . Furthermore, by only having hardware elements currently in use, energy expenditure can be reduced while obtaining high performance. Even with partial reconfiguration however, overhead is still a concern, and thus limiting reconfiguration as much as possible is of interest. One method is to consider floor plan layout of tasks at the design-level [76], so as to share resources of similar tasks as much as possible, reducing the amount of reconfiguration necessary. Others focus on scheduling tasks so that minimal reconfiguration is required [77].

While highly parallel algorithms perform well once mapped onto FPGA hardware, such parallelism is still limited by sequential portions of a task. Furthermore, FPGAs lack a standard programming model that developers can use to simultaneously design software and hardware. Different languages have been proposed and researched for this task [78, 79] but no clear standard has emerged, and developers still need to have knowledge of hardware in order to truly maximize performance. Soft-core processors bring familiar programmability to the table, but suffer from increased power usage when compared to heterogeneous multi-processors. Research into novel transistors [80, 81] aims to close this gap, but for the foreseeable future,

heterogeneous multi-processors will remain more efficient with power. Increased power consumption is often not an option for mobile devices, limiting FPGA use in the field; time to market is of equal concern, and the lack of generalized programming tools similar impedes FPGAs in this regard. As mobile devices take on a wider and more general application base, it will similarly become difficult to pick which tasks to implement in FPGA hardware without incurring prohibitive overhead costs from reprogramming. We now turn our attention to related work in speech recognition.

3.2 Speech Recognition Related Work

Speech recognition is a widely researched area. When selecting from the works of most relevance and importance to our research, we found that they fell into four general areas. We classified these areas as:

1) Dynamic Grammar Speech Recognition
2) Mobile and Real-Time Speech Recognition
3) Multi-Modal Speech Recognition
4) Contextually Partitioned Speech Recognition

We cover research on adapting speech through dynamic grammar models, based on different situations. In targeting mobile devices, we also want to focus on what resource constrained recognition work has been done. We discuss briefly a widely researched area in speech recognition – multi-modal recognition. Although not directly related, multi-modal is worth mentioning both because it has the ability to transform user interfaces, and has likely pulled attention from other research avenues, such as contextual partitioning. Finally, contextually partitioned speech recognition is what our research proposes, and we touch on the limited research available before presenting our own approach.

3.2.1 Dynamic Grammar Speech Recognition

While contextually partitioned recognition may not have been widely pursued, a similar approach of using dynamic grammars is more commonly documented. Dynamic grammar speech recognition focuses on adjusting the probabilities of the grammar model to ensure that only

words that fit the current context of speech will be returned by the recognizer. The Context-Aware Speech Interface System [15] (CASIS) from the University of Tokyo describes a natural language interface for a meeting room, leveraging sensors to detect physical context of the user in relation to the environment, along with characteristics such as temperature, brightness, and usage history. This information is stored in a context manager, which then applies weights to the grammar model, adjusting recognition probabilities to favor results that fit predicted behavior. Massachusetts Institute of Technology's (MIT) Intelligent Room [16] is similar in function to CASIS, but also describes a 'recognition forest' of many context-specific grammars that can be enabled or disabled based on contextual information. Grammars associated with specific devices or goals are deactivated if user interaction within those contexts is determined to be unlikely through sensory input, such as user position and history. Thus the recognition hypothesis space, as in the words that will be returned by the speech decoder, in the described forest is dynamic.

A moderately more mobile approach to dynamic grammars was implemented by the authors of [17]. Their work focuses on a speech driven electronic personal trainer that uses RFID tags to determine equipment usage. Language recognition accuracy was compared across grammars varying from full English dictation, to fitness specific and equipment specific grammars. Accuracy improvements were associated with size limited grammars through context specialization, so long as out of vocabulary instances were avoided. While this device is still tethered to RFID tags for providing context, the concept of environment classification is similar as to what our system would eventually implement as part of its context selection algorithm.

Dynamic grammar approaches, however, do not physically split the vocabulary, instead assigning probabilities to possible results to ensure that the final result fits a grammar model. This is an important distinction as while a dynamic grammar can greatly improve recognition accuracy, the corpus is still trained as a whole unit. Thus a dynamic grammar does not offer any of the training efficiencies we expect to see from context partitioned recognition, nor does it address the case of dynamic vocabularies. Furthermore, our research is more focused on the mobile environment, rather than constrained to a room. This is not to say that context partitioned recognition could not make use of dynamic grammars as well – it likely could, and would be an interesting avenue of further research.

3.2.2 Resource Constrained Speech Recognition

Devices such as mobile phones and PDAs add yet another challenging element to speech recognition – limitation of resources. When processing power, storage, and battery life are limited, different methods are required. These constraints are the primary reason why large vocabulary continuous-speech recognition has evaded the mobile market. Research in speech recognition for mobile devices, for our purposes, generally falls into one of two groups – embedded or network recognition [82]. Embedded recognition is where the device performs the entire recognition process itself, and experiences all the challenges noted previously. Most of the advances in this field come from algorithmic enhancements or architectural design, such as limiting memory bottlenecks or performing recognition in hardware. Network recognition meanwhile offloads either all of the recognition process (true network recognition), or does so partially (distributed recognition) [83]. This avoids many of the constraints of the devices themselves, but introduces the need for consistent network connectivity, and the possibility of signal corruption over the network.

For embedded recognition, emphasis has lately been placed on performing hardware recognition [84]. A primary constraint for this approach is vocabulary size – limited to around 1000 words – along with flexibility and cost [10]. Conversely, hardware recognition provides real-time recognition with low power expenditure. The flexibility issue is an important one however; some propose FPGA [85] or flash memory [86] implementations that would be reprogrammable with different vocabularies perhaps over a network [86], requiring a network connection to always be present. Either way, this would incur a reprogramming cost, which would limit fine-grained switching. The remote processing concept meanwhile has gathered considerable strength over recent years. Both Google and AT&T have released, or are releasing, cloud computing recognition applications for the iPhone [87, 88]. For situations where accurate, real-time recognition is required, and a network connection is not present or desired, these remote solutions are not feasible. Using remote speech to search Google is different from using speech as an interface to a device; the latter is much less forgiving with delays and mistakes. Local speech recognition solutions that are efficient and make use of well known tools, like SPHINX, are valuable for such markets.

3.2.3 Multi-Modal Speech Recognition

Multi-Modal speech recognition has been a hotbed of research in recent years [89-91]. While not directly related to our work, we think it likely that the emphasis placed on multi-modal recognition may explain why approaches such as our contextual partitioning have not been considered. Multi-Modal refers to systems that receive multiple modes of input from a user, with one mode being speech itself. Other common inputs include video for lip reading [92], gesture recognition [93], or manual input through a user interface, such as a touch screen [89]. A major draw for this kind of research comes from the need for robust speech recognition, especially in noisy and unpredictable environments. Whereas speech is plagued by distortion when noise is introduced, inputs such as lip reading will remain consistent. This is a powerful draw, and could usher in a whole new range of powerful user interfaces. Multi-modal recognition was a major factor in CMU's decision to begin development on SPHINX 4, so as to provide a recognition system better able to handle multi-modal inputs [26].

Some seem to suggest that input modes must be explicit interaction between user and device – thus our contextual inputs may not quality as a different mode of input, and our system is not necessarily 'multi-modal' as we describe it. Regardless, contextual partitioning will provide a strong basis for speech only recognition systems, that could then be extended into multi-modal recognition. While video processing necessary for lip reading and head tracking may not fit within the confines of mobile speech processing, other forms might fare well – a concept worth pursuing in the future. Environment classification [94], by using a variety of sensors, would likely play a role. Studies in [18] and [19] discuss using sensors along with electronic calendars to perceive whether the user should be interrupted by a phone call or message. The device is then able to adjust its ringer volume, relay owner status to callers, and allow callers to specify the urgency of their message to override anti-interruption measures. Multi-modal recognition deserves the attention it receives, and our proposed architecture will fit comfortably with it.

3.2.4 Contextually Partitioned Speech Recognition

We define contextual partitioning to mean that the vocabulary of a system is divided into multiple physically separate acoustic models to be processed by multiple independent recognizers. This is the foundation of our research, and to our knowledge has not been widely pursued. One project however, the Virtual Intelligent CO-driver (VICO) project out of

Europe[20], did propose contextual partitioning. VICO's goal was to develop a speech driven interface that would be built into vehicles to provide the driver with information, particularly for navigation. In order to process a large vocabulary of points of interest (hotels, restaurants, etc.) and navigation commands over a wide geographic area, the proposed architecture called for multiple speech recognizers that would divide the vocabulary based on commands and geographic regions. These recognizers each generate an output, and a top level algorithm would choose an output based on maximum-likelihood. Unfortunately the project appears to have dissolved, with the latest paper found released in 2005 discussing a first prototype. The project posted promising results, but did so using unusual speech recognition models rather than using well known and typical phoneme, tri-phone, or whole word units. The second phase was supposed to address these areas, to fully define the advantages of the system, but this second phase does not appear to have ever materialized. The project website itself (www.vico-project.org) has fallen into disrepair. Our best guess is that the project was discontinued, and does not appear to be picked up by another group.

Despite VICO's demise, we believe continuing work on contextually partitioned architectures in our research is valuable because we are considering it in a different market. VICO hoped to leverage CPs as a means to improve accuracy. While we aim for this goal as well, we also are interested in the possibility of using CP as a means to facilitate speech recognition on mobile devices. The capabilities of a system built into a car, in terms of processing power and size, are much more flexible than the platforms we are considering. VICO was also implemented using a speaker-independent corpus, rather than personalizing the database. While this has the benefit of not requiring users to train their own database, speaker-dependent recognition, if well trained, is superior in accuracy due to its ability to tailor the acoustic model on a particular user [12]. Our focus instead will be on a speaker-dependent corpus, concentrating on accuracy and training time, while also reducing decoding time so as to enable larger vocabulary real-time recognition on mobile devices. Finally, our testing will present results using well known tri-phone and/or whole word models. Our research will show CP allows for more accurate recognition, faster decoding times, and training efficiencies not provided by dynamic grammar recognition. CP will not rely on remote processing or specialized hardware otherwise common in the mobile market. Our solution will prove to be an effective speech-only solution that could be extended in the

future with multi-modal inputs, as multi-modal research matures. We'll now discuss our approach towards meeting these goals.

4 Problem Statement

Our goal in this research is to demonstrate the viability of a contextually partitioned architecture through an investigation of an example. Our target market is mobile devices, due to their growing pervasiveness and computational restrictions, as well as their ability to be personalized to the user. In order to define our contextual partitions, we created a sample user personality, whose occupation is a Northern Virginia based real-estate agent and is also a fan of the Washington Nationals MLB team. We use these contexts (occupation and hobby) to develop varying levels of contextual partitions.

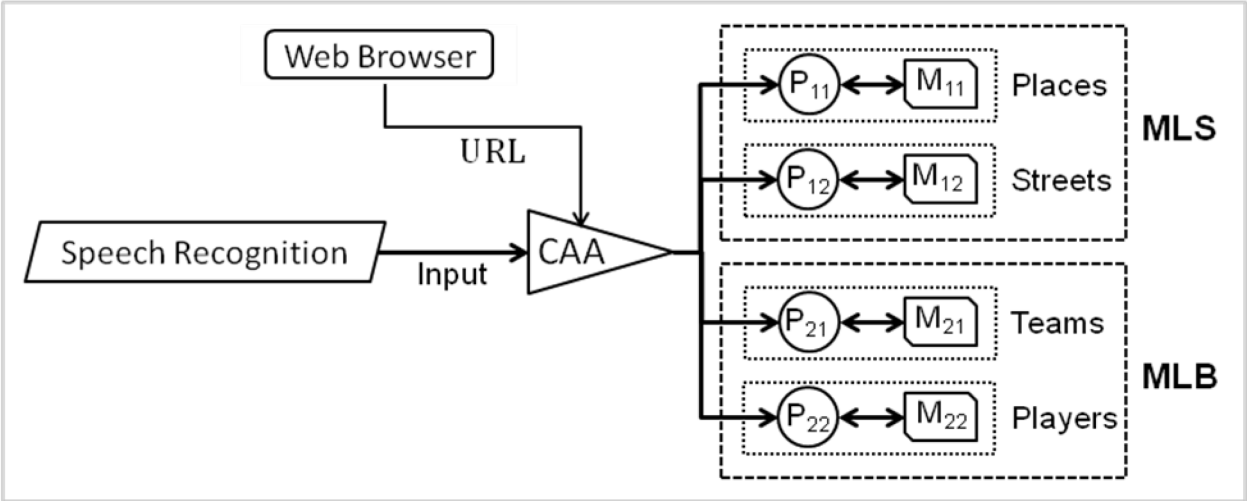

Figure 9: Realtor/Baseball Fan CP

Figure 9 above depicts a version of Figure 1 specialized for our example. We chose speech recognition as our target application, leveraging these partitions to divide a vocabulary contextually. Speech recognition was chosen because vocabulary size represents a primary bottleneck [95-97] for mobile devices, and it represents a powerful tool for future user interfaces. As Figure 9 depicts, our speech recognition is contextually divided in two ways: first coarsely into MLS and MLB, second into sub-contexts of those two partitions. The appropriate partition is selected by contextual information passed from the web browser. The results of this work form the foundation for contexts to eventually be related to interactions with websites, including which portion of a page the user is viewing and where the cursor may be. Figure 9 shows the contextual information as being the URL of the website the user is on, with a presumption that

other inputs can be used as well, including mouse tracking and user location (home, work, etc.). Reiterating our preliminary goals from earlier, our research will:

1) Explore Contextual Partitioning, both coarse-grained and fine-grained, within Speech Recognition to:
    a. Improve the Error/Training (ET) Curve
    b. Provide robustness for dynamic vocabularies
    c. Leverage greater numbers of PEs, maximizing decoding speedup
2) Define challenges for future work in Contextual Partitioning

Before performing experimentation, we needed to complete two more steps to further flesh out these goals. First, we had to create a vocabulary to partition and test with. Second, we needed to determine different levels of partitions within that vocabulary.

4.1 Vocabulary Definition

Given our sample personality – a realtor and baseball fan – we now had to determine a vocabulary that might be used to navigate a web browser through those contexts. We chose to focus on one frequently used task for each of these contexts. For 'realtor' we chose navigation of the Multiple Listing Service. For 'baseball fan' we chose navigation of the Major League Baseball website.

The MLS is a database where realtors post homes clients are selling, and search for listings other clients may want to buy. Clients themselves often have limited access to the MLS as well, and many real-estate firms are allowing viewing of MLS listings through web sites. According to www.realtor.org, "more than half of recent buyers used MLS web sites in their search"[98]. Thus, we consider MLS navigation to be a critical task for a real-estate agent, and find it likely a significant portion of an agent's web browsing would be spent navigating MLS websites. In order to generate a vocabulary for such navigation, we visited a MLS website (www.mls.com) and made note of different commands and words that would be useful. Commands for selecting Virginia, and areas of Northern Virginia were included, as were words used for creating,

describing, and searching listings, such as number of bedrooms/bathrooms, home architecture types, phone numbers, and prices.

Sports meanwhile are followed by a wide variety of people from all different walks of life. The MLB website is popular; at the time of this writing, entering into the World Series, Alexa ranks mlb.com 74th in terms of traffic of all websites[99]. We used the MLB website as a template for commands one would use to navigate, including commands for accessing each of the team websites either by team name or city name, the standings and scoreboard pages, and the player search and stats pages. Given our Northern Virginia based realtor personality, we chose our user to follow nearest team – the Washington Nationals – including all players on their active roster in our vocabulary.

Once compiled, we had a vocabulary of 337 words, which can be found in Appendix A. We now move on to consider how this vocabulary can be contextually partitioned.

4.2 Contextual Partition Definition

The most obvious contextual division of our vocabulary, is to split the MLB vocabulary from the MLS. This coarse-grained contextual partitioning is one case to consider.

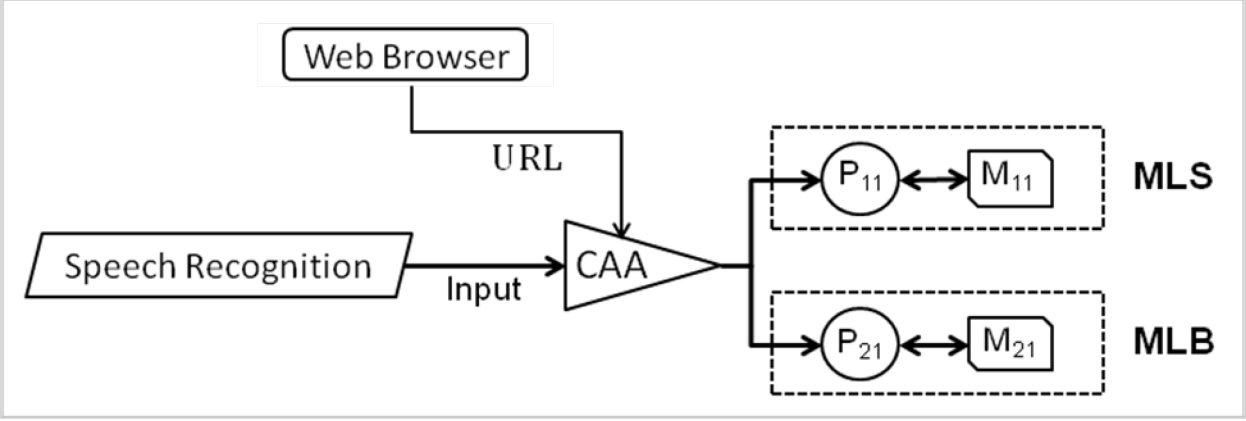

Figure 10: Realtor/Baseball Fan Coarse-Grained CP

But there are finer divisions within the contexts themselves, we term these sub-contexts. Examining the vocabulary specific to MLB and MLS respectively, we find several groupings of words that fall under definitive classifications. We use these groups as our sub-contexts, listed below:

1. MLB
    a. Cities: List of cities MLB teams are located in
    b. Names: Players on the active roster of the Washington Nationals
    c. Teams: Teams in MLB
    d. Navigation: Commands for navigating the MLB website
2. MLS
    a. Area: Regions of Northern Virginia
    b. Description: Descriptive words used in advertising homes
    c. Navigation: Commands for navigating an MLS website or program

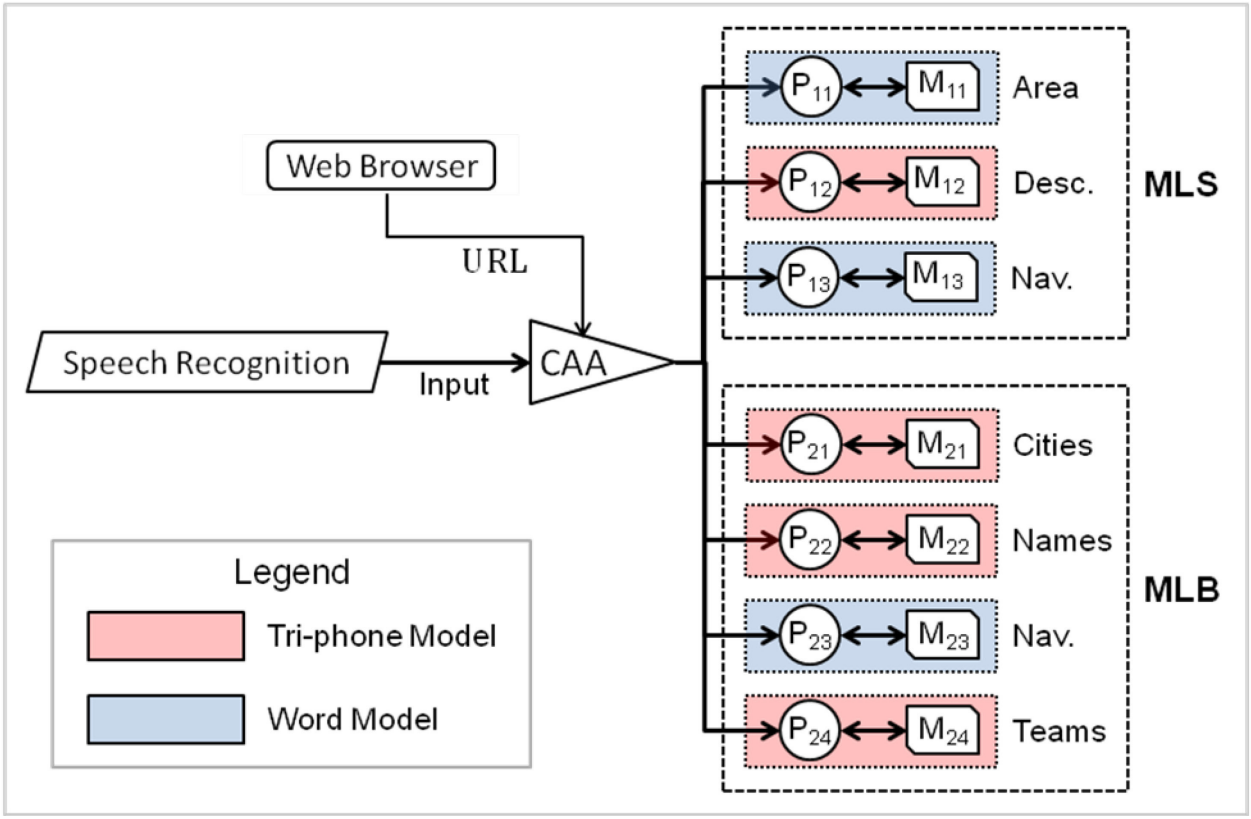

Figure 11: Realtor/Baseball Fan Fine-Grained CP, Hybrid Models

Figure 11 depicts our architecture, once specialized with both our contexts and sub-contexts. Further division of our vocabulary introduces several new considerations. First, sub-contextual division leads to assignments of smaller vocabularies. If the vocabularies for such sub-contexts are small enough, it may be possible to use whole word models. This could ultimately result in a recognition system that uses some partitions trained with tri-phone models, and others trained

with word models, as seen in Figure 11. This would be a wholly unique artifact to our architecture, and its significance is something we will consider in our experimentation.

Another point to consider in our partitions is a case of overlapping vocabulary. While our MLS and MLB contexts are largely mutually exclusive, they do share some words in common – notably numbers, both for player numbers in MLB and phone numbers/listing IDs in MLS. A question we had to consider is how to handle this overlap in vocabulary. We see two methods of handling this situation. First, it would be possible to replicate the overlapping entries across applicable partitions. With coarse-grained partitioning, this is fairly easy; entries that are applicable in both MLS and MLB appear in both MLS and MLB, resulting in our coarse-grained architecture presented earlier in Figure 10. For fine-grained partitioning though, entries may have to be replicated over many sub-contexts, resulting in greater overhead – a less promising situation. Alternatively, one could implement a separate partition for common entries – a "general" partition (GP).

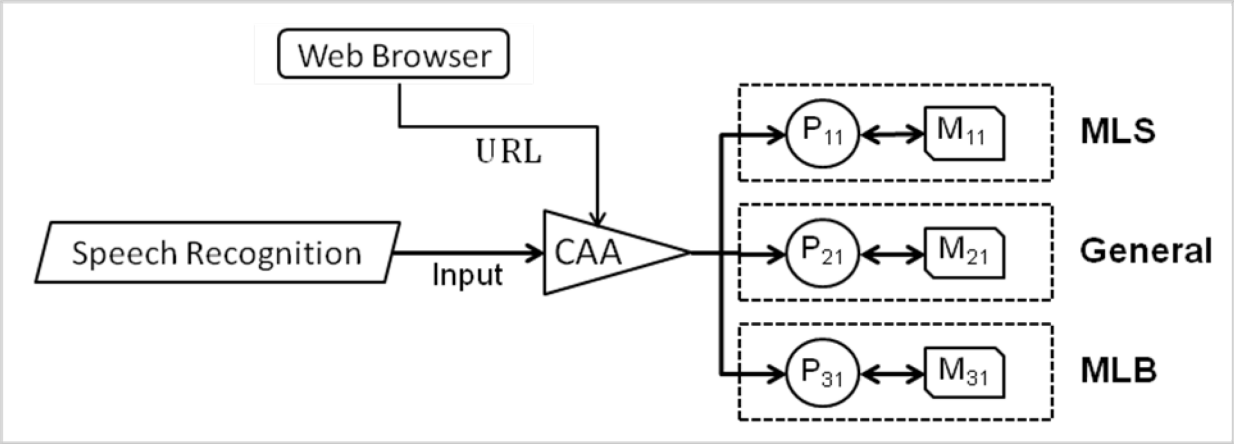

Figure 12: Realtor/Baseball Fan Coarse-Grained CP with GP

Figure 12 and Figure 13 show our architecture layout when a GP is included for coarse-grained and fine-grained partitioning respectively. We believe the GP solution to be more feasible for fine-grained partitioning because it prevents vocabulary duplication across many sub-contexts. Thus, our experimentation will compare partitioning methods including coarse-grained with and without GPs, and fine-grained with a GP. Having decided on a test vocabulary, and discussing several partitioning schemes, we will now crystallize our experimental goals.

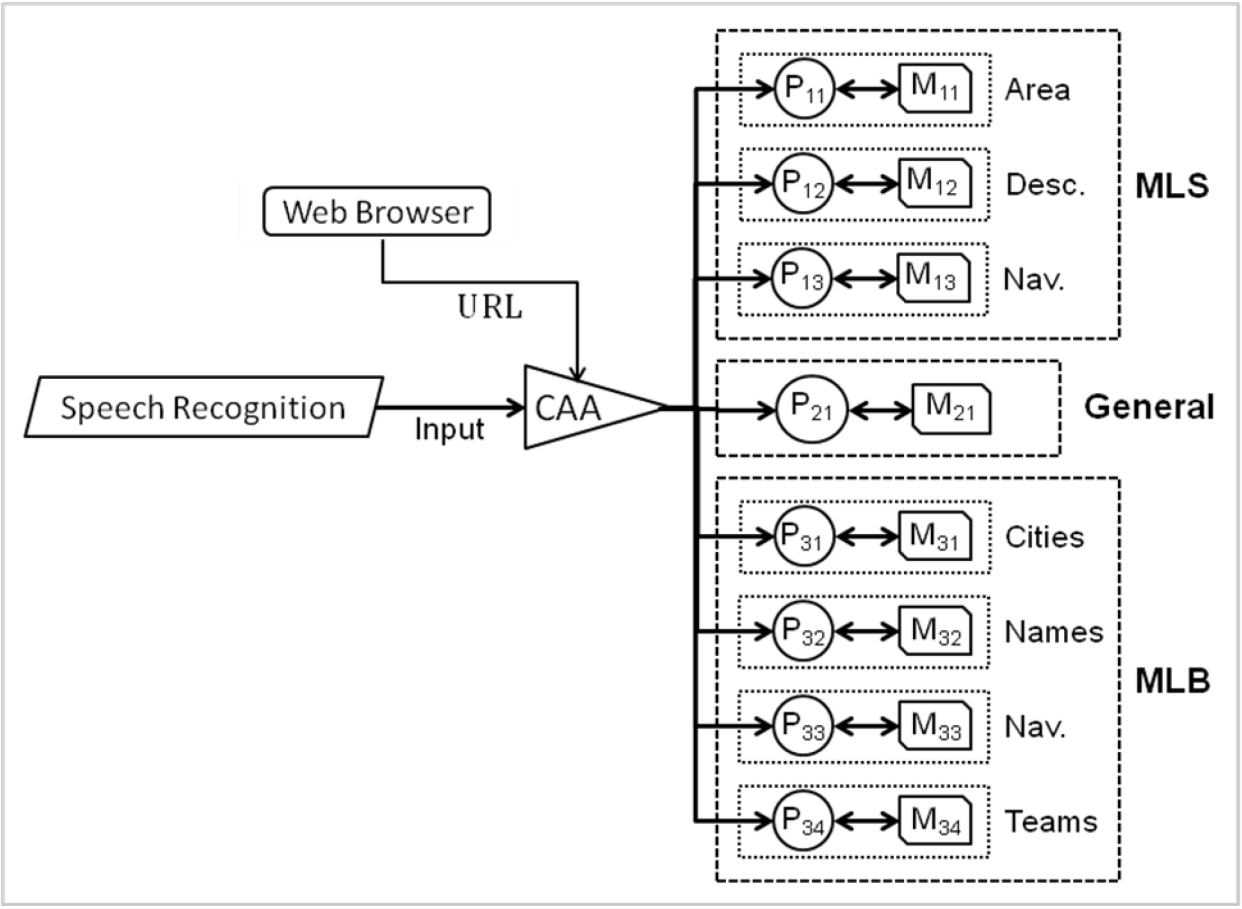

Figure 13: Realtor/Baseball Fan Fine-Grained CP with GP

4.3 Experimental Goals

Our exploration of contextual partitions within our sample vocabulary led us to a number of different options. Our experimentation will focus on the following partitioning methods:

1) Unified (No CP), with Tri-phone Models
2) Coarse-Grained CP without GP, with Tri-phone Models
3) Coarse-Grained CP with GP, with Tri-phone Models
4) Fine-Grained CP with GP, with Hybrid Models

Each partitioning scheme will be evaluated over our previously mentioned performance metrics:

1) Error/Training (ET) Curve
2) Robustness with Dynamic Vocabularies
3) Maximize decoding speedup through leveraging more PEs

Our interest in the ET curve stems from the desire to limit user training required to use the system, as well as the hope that by reducing this training, less training would be required to adapt a speaker-independent corpus if one were available. Thus, we will measure for each increase in training, the change in accuracy of the system for each partitioning scheme. Given our interest in personalizing the architecture, it is likely that the systems vocabulary will change over time. Thus, we are interested in how each of the partitions responds to changes in the vocabulary – specifically how much training is required to add new words, and what cost in accuracy is associated with removal of entries. Finally, we are interested in making a comparison between contextual level parallelism, generated through contextual partitioning, and thread level parallelism across an increasing number of PEs. We will now describe our experimental setup.

5 Experimental Setup

We review our experimental setup by first discussing selection of speech recognition software. Then an overview of our particular software implementation is provided, including technical details about equipment used. Next we confer how we will measure our chosen performance metrics. Finally, we discuss the details of each experiment individually.

5.1 Speech Recognition Suite Selection

One of the first decisions to be made, when planning our experimentation, was what speech decoder to use for our tests. Some options, such as Microsoft's Hidden Markov Model Toolkit (HTK) and Dragon's NaturallySpeaking, are proprietary, while others, most notably Julius and Carnegie Melon University's SPHINX are open source. The benefits of using open source software narrowed our candidates to these two. Both are top of their class and would make a fine choice. Julius has been focused primarily on the Japanese language however. SPHINX meanwhile is widely used [36, 100, 101] for many different languages including English, and has established credibility as a formerly DARPA funded project. SPHINX is also supported by a range of utilities provided by CMU, such as pronunciation dictionaries and the Statistical Language Modeling (SLM) toolkit. Thus, we decided to use SPHINX for this research. The SPHINX decoder comes in several different varieties. Of most interest to us is PocketSPHINX, a light-weight decoder meant for embedded systems, which is designed to process smaller vocabularies in real-time.

PocketSPHINX is a HMM-based decoder, with N-gram support. It allows for direct manipulation of many HMM parameters: number of states, model type (triphone, word, etc.), state topology (either Bakis or Linear), language weight, and number of senones. As with any HMM-based decoder, PocketSPHINX requires training to build its models. CMU provides a utility, SPHINXTrain, for feature extraction of raw audio data and model generation. PocketSPHINX and SPHINXTrain depend on a number of files to define the training and recognition process:

1. Train/Decode Configurations: Files that contain parameters for SPHINXTrain and PocketSPHINX.
2. Dictionary: Maps the system's vocabulary to a transcription of speech units. Typically the units are phonemes, but others, such as whole word units, can also be used.
3. Unit list: Lists all speech units to build models from (e.g. whole words or phonemes). Phonemes are listed if tri-phones are to be used – the system generates its list of tri-phones based on the dictionary automatically.
4. Grammar Model: Probabilities of N-word combinations represented by an N-gram model.
5. Training Fileids/Transcriptions: List of raw speech files to be trained with, and transcriptions of the contents of each file.
6. Test Fileids/Transcriptions: List of raw speech files to generate decoding hypotheses for using the models generated in training, and transcriptions of the contents of each file used to determine accuracy.

We'll now describe our implementation of PocketSPHINX, and the above variables.

5.2 SPHINX Environment Creation

Having decided on a vocabulary, our first step is to create the lexical model, or dictionary for each partition. All applicable vocabulary words are added to a dictionary file for each contextual partition, where each entry is broken into a series of speech units. We used phoneme units for larger partitions, and word units for smaller partitions (about 50 words or less). In order to use phonemes, one must know the pronunciation of each word in the vocabulary. CMU provides a lexicon tool (www.speech.cs.cmu.edu/tools/lextool) containing pronunciations for over 125,000 words using a standard set of 40 phonemes, which we used. Many words have multiple proper pronunciations however, as well as slang forms (e.g. dropping the 'g' in '-ing'), and it is important to include pronunciations that are actually used by the user in training and testing. Since we are focusing on personalized, speaker-dependent devices, we can limit our pronunciations to only those used by our speaker, a luxury not afforded by speaker-independent corpuses.

The next step was to record samples to both train our acoustical models for each partition, and to test with during the decoding process. Recall our focus is isolated word recognition, thus the majority of our samples contain only one vocabulary entry each (exceptions being two-word names such as "New York"). Each sample was approximately 5 seconds in length with 1-2 seconds of entering and trailing silence. All samples were recorded by the same speaker using a Cyber Acoustics AC-201 headset microphone with a built-in soundcard of a Dell D800 laptop under quiet conditions. Samples were stored as mono files with 16,000 Hz sampling rate and 32-bit PCM quantization. Test samples were separate and unique from training samples.

At this point, the acoustic models can be trained. For our experiments, most of the variables of SPHINXTrain were left at their defaults. Training was performed using CMU's SPHINXTrain version 1.0 Revision 8978 with mel frequency-warped cepstrum (MFC) feature extraction, with 13 coefficients (MFCCs). 1000 tied states (senones) were used. 5-state semi-continuous Bakis topology HMMs were generated for PocketSPHINX to decode from, the default. Some of our experiments adjusted a few variables, notably model type for whole word HMMs. Specific modifications will be discussed in each experiment. For evaluating each partition's performance, we decoded our test samples using PocketSPHINX. To supplement this process, a language model must be provided. Since our focus is on isolated word recognition, our language model is simply a list of entries in the vocabulary. This will give us a baseline recognition performance, from which further research can explore the benefits of more complex language models, particularly for continuous speech recognition. Decoding was performed using CMU's PocketSPHINX version 0.4.1 Revision 7200. Some experiments adjusted a word insertion likelihood; applicable tests are noted in their respective sections. Both training and decoding were performed on a 2.67 GHz Core 2 Duo Desktop with 2 GB of DDR2 memory running Fedora 9 32-bit. We'll now cover how we measured our performance metrics.

5.3 Performance Measurement

For each partitioning method, we are interested in measuring:

1) Accuracy versus Training Time (ET curve)
2) Robustness to change in vocabulary
    a. Change in number of errors when entries are removed
    b. Change in number of errors when entries are added
    c. Training required to recognize new entries
3) Decoding speedup over sequential and TLP

In order to determine accuracy we have a recording, or test sample, for each word in our vocabulary. We evaluate accuracy by running applicable test samples for each partition through PocketSPHINX, which then returns the accuracy by comparing each hypothesis against the word actually contained in the test sample (provided by our fileid and transcription files). PocketSPHINX returns two error values: sentence error and word error. Sentence error is the number of sample inputs that contain errors, where each sample can be one word or several. Word error is the number of words in a sample set that had errors. Thus, a sample containing multiple words that had multiple errors will count as a single sentence error, but multiple word errors. The error rates returned by PocketSPHINX are the sum of three different types of errors. They are insertion, deletion, and substitution errors.

```
***       SEVENTY  (Sample)
CITY      T        (Hypothesis)

Words: 1  Correct: 0  Errors: 2  Error: 200%
```

Figure 14: Decoding Result Example

Insertion errors are where more words are hypothesized than were uttered, while deletion is the opposite. A substitution error occurs when an incorrect word is hypothesized for an utterance. In Figure 14, ‘CITY’ is an insertion error, for there was no word in the sample to correspond with it, while ‘T’ is a substitution error because it was hypothesized for the actual word ‘SEVENTY’. The error count for each of these error types is summed and then divided over the total number of words to determine the error rate of the decoding process. It is therefore possible, due to insertion and deletion errors, to have an error rate greater than 100%. When we refer to accuracy, we refer to the word error returned by PocketSPHINX, as we are interested in the total accuracy of each word hypothesized by the system.

We are interested in improving accuracy, thus training is an iterative process, where we train with samples for each of the words in a partition, and then add samples to those words that are misrecognized. This presents a first variable for measuring training cost: number of samples. The sample count exhibits two different costs to the user. First, it represents how many training samples the user had to personally record. Thus, reducing the number of samples reduces the amount of recording time required. Furthermore, training requires that we keep feature files generated from each training sample. The more training samples we have, the more feature files must be stored. While these tend to be very small, it still is a cost. A second variable for training evaluation is also related to this iterative training process – training time. Each training iteration takes a certain amount of time to complete, during which the device is busy at the cost of the user. Reducing the amount of training time saves the user time, and is thus important. Training time was calculated by running the UNIX 'time' command over the training process and summing the user and system portions returned. Training time for a partition is recorded as the sum of time spent training each iteration plus the amount of time spent recording samples. Recording time was calculated as 5 seconds multiplied by the number of samples used, as each sample was approximately 5 seconds in length.

Using the above three metrics, we perform two experiments comparing coarse-grained and fine-grained contextual partitioning against a unified system. We generate a plot of accuracy versus number of samples, and a plot of accuracy versus training time, for each partitioning scheme, as a basis of comparison. Following our ET curve measurements, we performed a "dynamic corpus" test, to measure robustness of partitions to changes in the vocabulary. We removed a set of vocabulary words, and then recorded the change in the number of errors for each affected partition. We then added a similarly sized set of words to each of those partitions, retrained until those new words could be successfully recognized, and recorded the amount of training time and the overall accuracy of the retrained system.

Following our experimentation, the final metric, decoding speedup, is generalized using trends from previous works. We compare the speedup provided by TLP against CP for increasing numbers of cores. Although we did not perform testing on an embedded system, our generalizations reflect sound intuition given existing literature, and provide a strong basis for further work and potential implementation on a mobile platform. The details of each experiment will now be discussed.

5.4 Coarse-Grained Partitioning

Our first experiment compares accuracy of isolated word recognition per training time and training samples for a unified case versus our coarse-grained contextually partitioned vocabulary of MLB, MLS, and different variations of General. Three test cases were compared:

1. Unified Processing: MLB, MLS, and General vocabulary combined with no partitioning
2. Contextual Processing without GP: MLB and MLS split into separate partitions, general vocabulary replicated into both MLB and MLS.
3. Contextual Processing with GP: MLB and MLS split into separate partitions, separate General partition.

Each of the test cases began with two samples per vocabulary entry in each partition. Since these partitions are using tri-phone units, a bad training sample for one word that models a tri-phone incorrectly can affect any word that also uses that tri-phone, causing multiple errors. We account for this by starting with two samples per word, so as to attempt to balance such a situation. The acoustic models were trained with this base sample set, and the word error rate was recorded. Samples were then adjusted for misrecognized words. Deletion and substitution errors were handled by adding samples to the missed entry. If adding samples did not remove an insertion error, removing samples from the inserted word was also attempted. Samples were added one at a time for the most part, except for particularly stubborn words where they were added two at a time instead.

Partition	# Words	HMM Unit
Unified	337	Tri-phones
MLB	164	Tri-phones
MLS	144	Tri-phones
General	29	Tri-phones
MLB+Gen	193	Tri-phones
MLS+Gen	173	Tri-phones

Table 2: Coarse-Grained Partitions

A quick word about terminology: we uses '+' to denote corpuses that have been combined into the same partition and '/' to represent corpuses that have been separated into different partitions. Thus, MLB+MLS+Gen represents our unified corpus. MLB/MLS/Gen represents our coarse-grained partitioned architecture with a separate GP. MLB+Gen/MLS+Gen then refers to a

coarse-grained partitioned architecture with MLB and MLS in separate partitions and the General vocabulary duplicated into both.

5.5 Fine-Grained Partitioning

Our coarse-grained analysis demonstrates only one level of contextual division – dividing our vocabulary into its respective MLB and MLS parts. To test finer contextual partitioning, we took our MLB and MLS contextual partitions and split them into respective sub-contexts. We then compared the ET curve between coarse-grained and fine-grained partitioning. MLB's vocabulary was split into the sub-contexts of Names, Teams, Cities, and Navigation; MLS's vocabulary was split into the sub-contexts of Area, Description, and Navigation. Performing this division resulted in partitions with small vocabularies. Due to this, it became possible to train small vocabulary partitions with whole word HMMs, while continuing to train larger vocabulary partitions with tri-phones. Since whole word HMMs are supposed to work well with small vocabularies, it would be of value to determine if their use in our contextual division would be beneficial.

5.5.1 Word Model Configuration

To use whole word models, several adjustments had to be made to SPHINX variables. Primarily, we had to choose what acoustic model type to use. SPHINX generates three types of models:

1) Context-independent: models are made for each speech unit
2) Untied context-dependent: models are made for each 3-tuple of speech units, where each speech unit is related to its neighbors (for phoneme units each tuple is a tri-phone, thus each tri-phone is modeled), and similar HMM states are not grouped.
3) Tied context-dependent: models are made for each 3-tuple, where each speech unit is related to its neighbors, and similar HMM states are grouped together

SPHINX does not distinguish between when one is using phoneme or word units in the dictionary. Therefore, even when using word models, SPHINX generates these three models by

default. Context-dependent models with whole word units seem strange, since words are now modeled by individual units and their neighbors thus are other words – a context-dependent acoustic model would appear to take on a role similar to the grammar model. Furthermore, nearly all of our samples contain a single word, thus neighbors of each word will usually be silence entries. Therefore, one might expect we would use the context-independent models. We found however, that using context-independent models incurred higher error rates than context-dependent, even with whole word models. This seems unusual; we can only surmise that relating word entries to the silence entry helped the recognition. From an implementation perspective, we chose to select the model that performed best, which was the untied context-dependent.

In performing this test, we noted that when using whole word HMMs the number of insertion errors tended to dramatically increase, particularly with small words (e.g. letters). In order to counteract this we found it necessary to adjust a parameter named the Word Insertion Probability (WIP). This parameter is the likelihood that the decoder will hypothesize a word for an utterance. By lowering it, it makes it less likely that a decoder will hypothesize multiple words for an utterance. Furthermore, it also makes it less likely that the decoder will hypothesize several words, if several words were actually uttered. While we only consider isolated word recognition in this research, we note that for continuous speech recognition this could become a problem. Therefore, we only used whole word models with partitions that contained a vocabulary with mutually exclusive entries, meaning that two words from the same vocabulary would not be uttered concurrently. This ensures that the decoder will not have to worry about consecutive words from a WIP adjusted partition, and we can adjust the WIP freely, even in continuous speech situations. Fortunately, adjusting the WIP is a decoder level parameter, and thus does not require retraining. Nevertheless, the minimal time spent adjusting WIP was included in our training time calculations, since it was a step that our phoneme based contexts did not have to endure. We now set about configuring our different partitions to be trained either as word or tri-phone trained.

5.5.2 Fine-Grained Partition Configuration

Partitions that we trained with whole word models started with a baseline case of using 1 sample per word. We adjusted samples largely in the same fashion as in our coarse-grained

testing, adding 1 sample at a time, but copious insertion errors were also handled by reducing the WIP. Because we trained our whole word HMMs with a baseline of 1 sample per word, in order to have a fair comparison, we also trained our tri-phone models, including the MLB and MLS partitions, with 1 sample per word for this test. Strictly for comparison sake, we also attempted to train MLB and MLS with whole word HMMs. This however, is not recommended as it performed very poorly.

Partition	# Words	HMM Unit	HMM Type	WIP
pMLB	164	Tri-phones	CD_tied	0.2
wMLB	164	Word	CD_untied	0.2
Team	30	Word	CD_untied	1 E -6
Name	48	Word	CD_untied	5 E -5
City	27	Word	CD_untied	1 E -9
MLB Nav	59	Tri-phones	CD_tied	0.2
pMLS	144	Tri-phones	CD_tied	0.2
wMLS	144	Word	CD_untied	0.2
pArea	88	Tri-phones	CD_tied	0.2
wArea	88	Word	CD_untied	1 E -7
Desc	28	Word	CD_untied	1 E -19
MLS Nav	34	Tri-phones	CD_tied	0.2

Table 3: Fine-Grained Partitions

In order to differentiate on the plots between the tri-phone trained and word trained coarse-grained partitions, we labeled them with preceding ‘p’ and ‘w’ respectively. For example, wMLS represents a word trained MLS partition. For MLB’s sub-contexts, all except the navigation partition were trained with whole words. MLB’s Navigation sub-context contains words that would likely follow one another, and a larger vocabulary count, and therefore was trained with tri-phones. MLS, in general, was a harder context to subdivide. Of its 144 words, 88 of them refer to various areas of Virginia (e.g. cities, counties, and communities). While it would be possible to divide these based on technical categories, such as census-designated areas, versus non-designated, versus counties and cities, such a division, though technically correct, would probably not be intuitive to the user or feasible for a selection algorithm to arbitrate between. Counties, cities, and census-designated areas are often used interchangeably in both language, and in mailing addresses, and firm definitions appear to be vague. As a result, we end up with a sub-context, Area, containing 88 words. Area is joined by two other sub-contexts: Description,

containing various descriptors used when advertising homes, and MLS Navigation, containing commands for navigating a MLS website. We are faced with a difficult question however. Is Area too large for whole word models? Furthermore, is it small enough that tri-phones will not function well either? In the end, we decided to test both cases, to see how each would fair. Once again we started with a base of 1 sample per word and show wMLS merely for comparison.

We now move on to discuss our final experiment covering situations where the vocabulary of the system may change – or a dynamic corpus.

5.6 Dynamic Corpus Comparison

In order to compare the effects of changing a corpus' vocabulary across our different partitions, we focused on the player names in the MLB vocabulary from the Washington Nationals active roster list. We compared the active roster list we obtained from the MLB website when experimentation began to the list after completing our coarse-grained and fine-grained testing. We found a total of 22 names in our corpus that were no longer valid, and 24 new names that needed to be added. Our dynamic corpus experiment compared how our partitions performed both when we removed the 22 names, and when we added the 24 names. We performed this experiment for the Names sub-context, MLB, MLB+Gen, and MLB+MLS+Gen.

For the removal test, we deleted all 22 names from the corpus, retrained the corpus, and then decoded each of our test samples for the remaining vocabulary words. For the addition test, we took our corpus from the deletion test, having removed all the old names, and added the 24 new names to it with 1 sample each. After retraining, we decoded once again with a test sample for each vocabulary word, including the new ones. The corpus was retrained, and the samples for the new words adjusted 1 sample at a time, until all of the new words were recognized with 0% error. Words that existed in the corpus previous to this experiment did not have samples added or removed, and their accuracy was ignored in the retraining. It seems likely that a user when adding words would not check the accuracy of the old words when retraining. Errors of that nature would instead likely show up in device usage – impeding the user's experience interacting with the device. Thus, for each of these tests, we measured the change in the number errors caused by the addition or removal or words, as well as the retraining time. To perform this

comparison we took the best run of each corpus from our previous experiments and trained it three times, recording the average error rate and training time. The removal test was also run three times and averaged, while the addition test was retrained until the corpus could be trained three times in a row and achieve 0% error rate for the new words.

6 Results

Here we present the results for each of the experiments outlined in our Experimental Setup section. We focus on two of our three performance metrics here: the ET curve and robustness, covering decoding speedup and scalability in the next section.

6.1 Coarse-Grained Partitioning

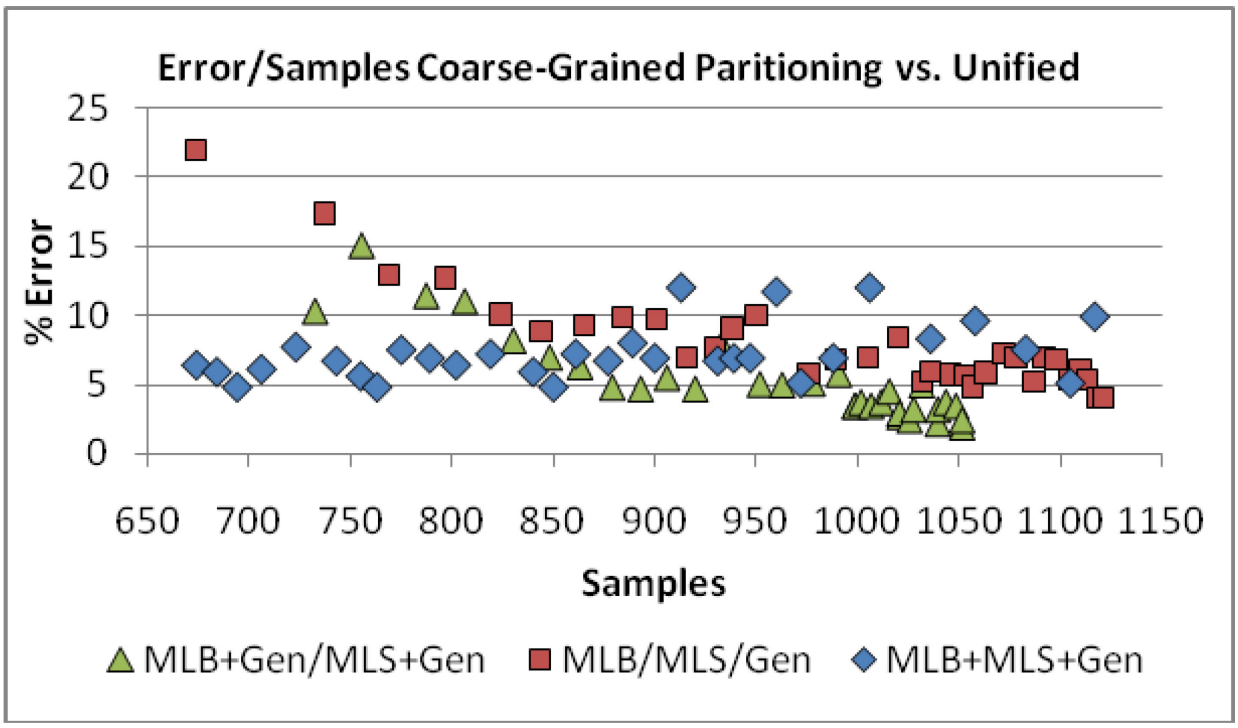

Figure 15: Error/Samples Coarse-Grained Partitioning vs. Unified

Figure 15 above compares accuracy to sample count for coarse grained, with and without a general partition, against a baseline unified, non-partitioned case. The baseline remained relatively flat throughout the sample range, perhaps worsening towards the end. The coarse-grained partition cases meanwhile have a definitive trend of improving with additional samples. Our best results came from not using a GP, instead replicating the general vocabulary into both the MLB and MLS partitions, represented by MLB+Gen/MLS+Gen. This approach quickly

converged and overtook the accuracy presented by our baseline case, achieving an error rate of 2.49%, while our baseline managed 4.8% - this represents a 51.9% improvement in accuracy. Coarse-grained partitioning with a GP saw only a minor improvement over the baseline, achieving an error rate of 4.05%, but after considerably more training than coarse-grained without GP. The unified case showed no signs of improvement; should it ever approach the accuracy provided by MLB+Gen/MLS+Gen it would appear to do so at considerably higher training cost.

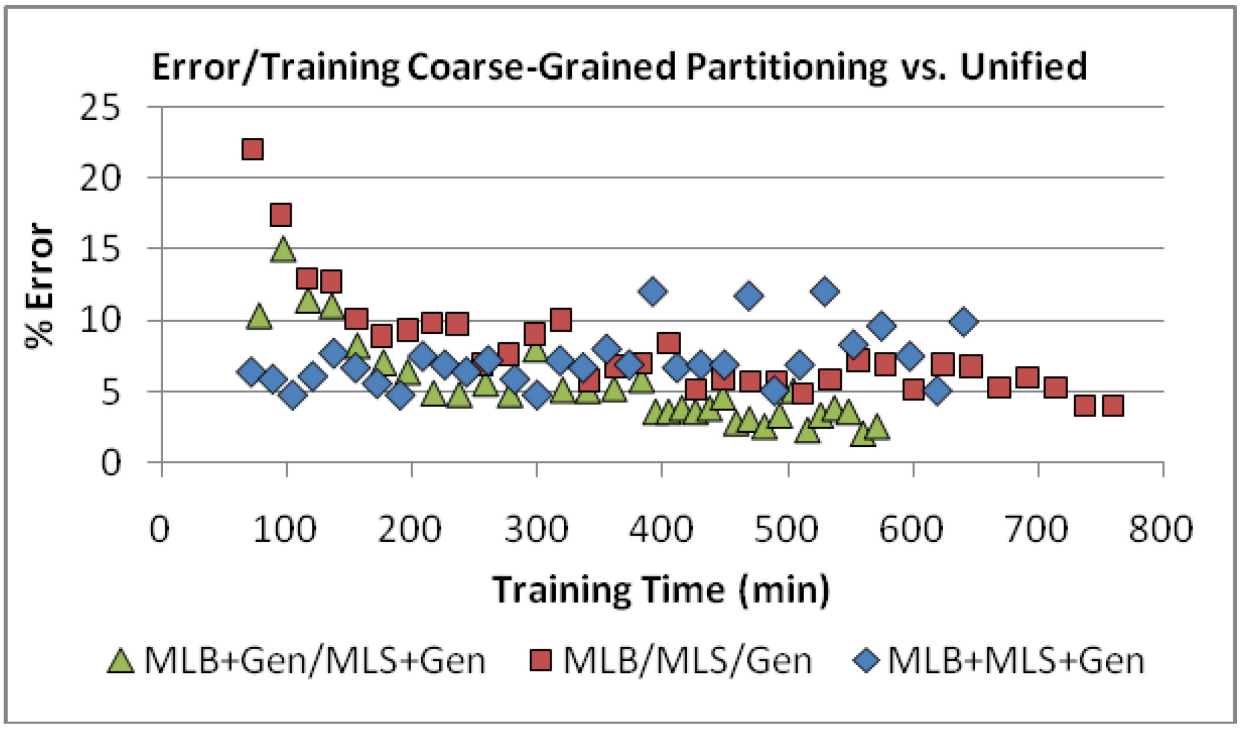

Figure 16: Error/Training Coarse-Grained Partitioning vs. Unified

Figure 16 meanwhile compares error rate across training time, and largely shows the same trends as our sample plot. While the baseline remains relatively flat across the training period, both coarse-grained cases have definite improvement with increased training. MLB+Gen/MLS+Gen is similarly strong in this comparison, overtaking the baseline quickly, while MLB/MLS/Gen does so at a considerably slower pace.

Our coarse-grained results provided increased accuracy per training, both in samples and time, fulfilling our first performance metric – the ET curve. They also served to fuel our hopes that we would see even greater improvements when leveraging fine-grained CP.

6.2 Fine-Grained Partitioning

Partition	# Words	HMM Unit	HMM Type	WIP
pMLB	164	Tri-phones	CD_tied	0.2
wMLB	164	Word	CD_untied	0.2
Team	30	Word	CD_untied	1 E -6
Name	48	Word	CD_untied	5 E -5
City	27	Word	CD_untied	1 E -9
MLB Nav	59	Tri-phones	CD_tied	0.2
pMLS	144	Tri-phones	CD_tied	0.2
wMLS	144	Word	CD_untied	0.2
pArea	88	Tri-phones	CD_tied	0.2
wArea	88	Word	CD_untied	1 E -7
Desc	28	Word	CD_untied	1 E -19
MLS Nav	34	Tri-phones	CD_tied	0.2

Table 4: Fine-Grained Partitions

Table 4 once again shows our partitions used for Fine-Grained testing along with their final WIP value used for decoding. Recall the WIP was adjusted (from a default of 0.2) to reduce insertion errors for our word trained sub-contexts. We found that our MLB and MLS partitions responded differently when divided into sub-contexts, thus we present our results for each in separate sections.

6.2.1 MLB Fine-Grained Partitioning

Examining Figure 17, it quickly becomes apparent how effective this division was for MLB. Figure 17 shows, at 242 samples, our sub-contexts achieved an error rate of just 1.83%. This represents an increase of accuracy of 92% when compared to tri-phone trained MLB (pMLB) at about the same sample count (244 samples). Even extending the training period well beyond our best marks with the sub-contexts does not allow pMLB to achieve the same accuracy.

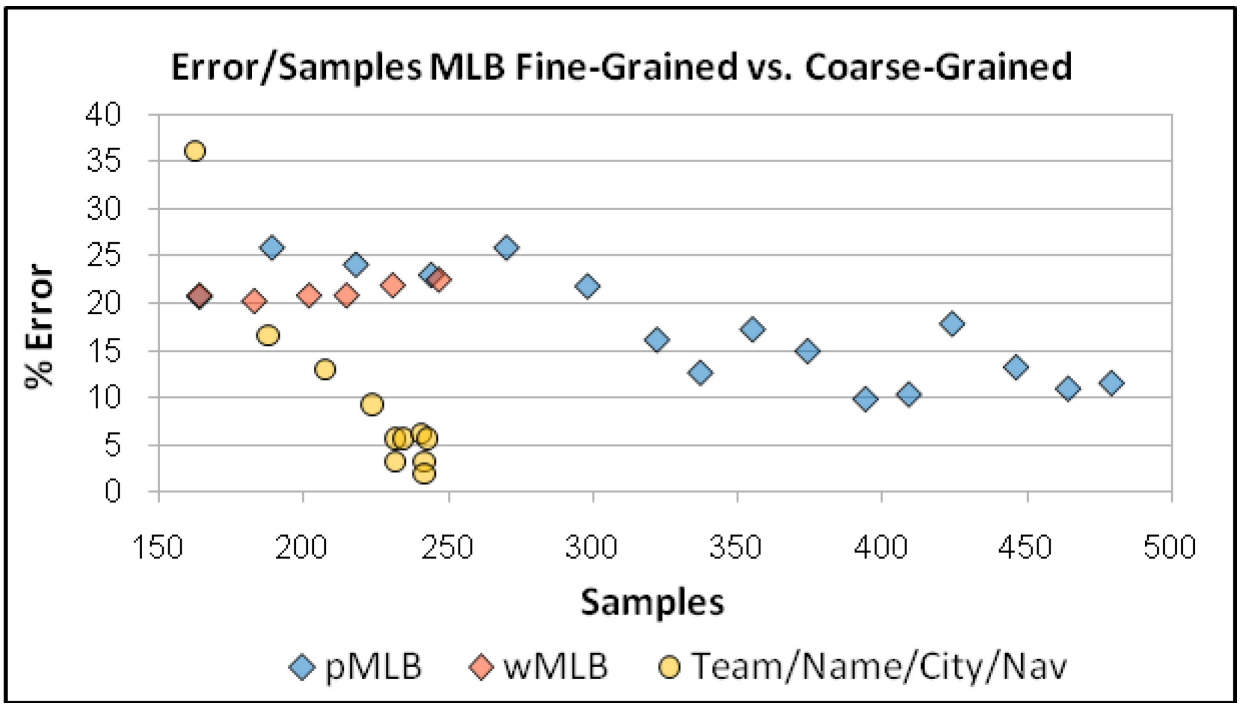

Figure 17: Error/Samples MLB Fine-Grained vs. Coarse-Grained

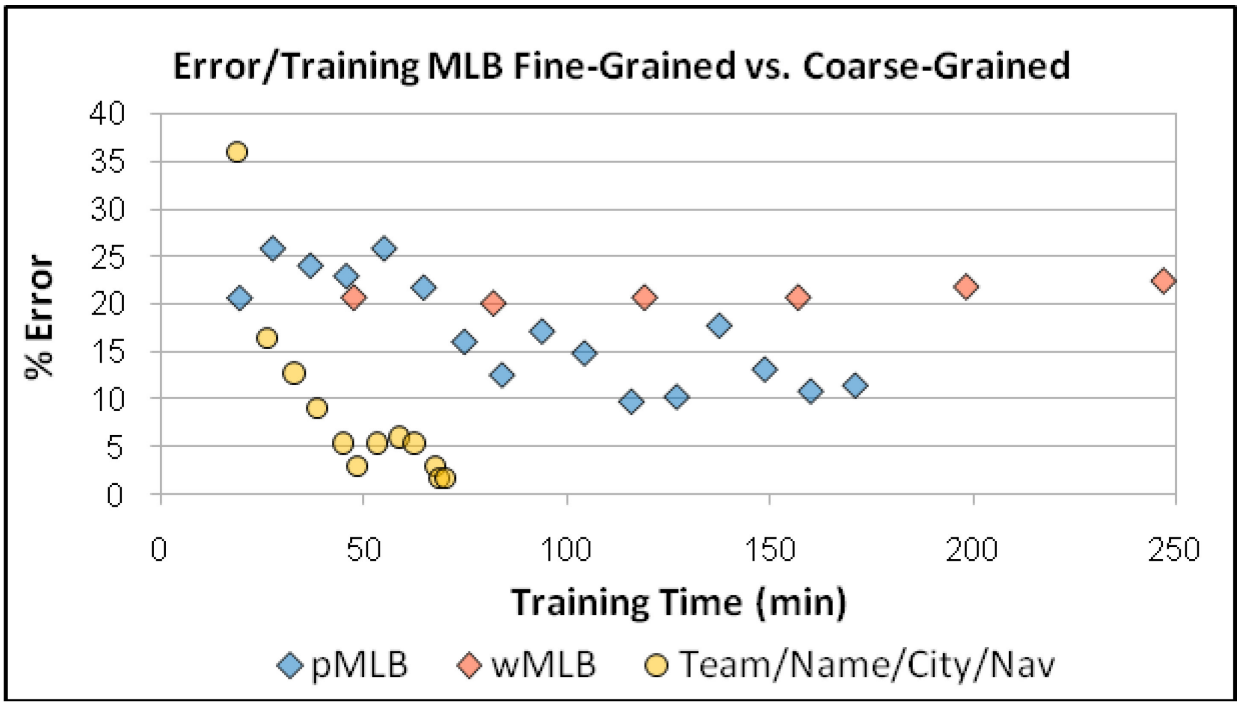

Figure 18: Error/Training MLB Fine-Grained vs. Coarse-Grained

One might, at first glance at Figure 17, think that we cut our training of whole word trained MLB (wMLB) short. Examining the training plot in Figure 18, however, shows that in actuality we spent longer training wMLB than any other model. This shows a notable limitation of whole

word models, which is already well known – they do not scale well with increases in vocabulary. With MLB's 164 words, it was taking 30-40 minutes per iteration to train wMLB. For our smaller vocabularies however, word trained HMMs performed quite well. Our fine-grained CP MLB achieved its error rate of 1.83% at 70 minutes of training – representing an accuracy improvement of 88.6% when compared pMLB at about the same training time (74.85 minutes).

6.2.2 MLS Fine-Grained Partitioning

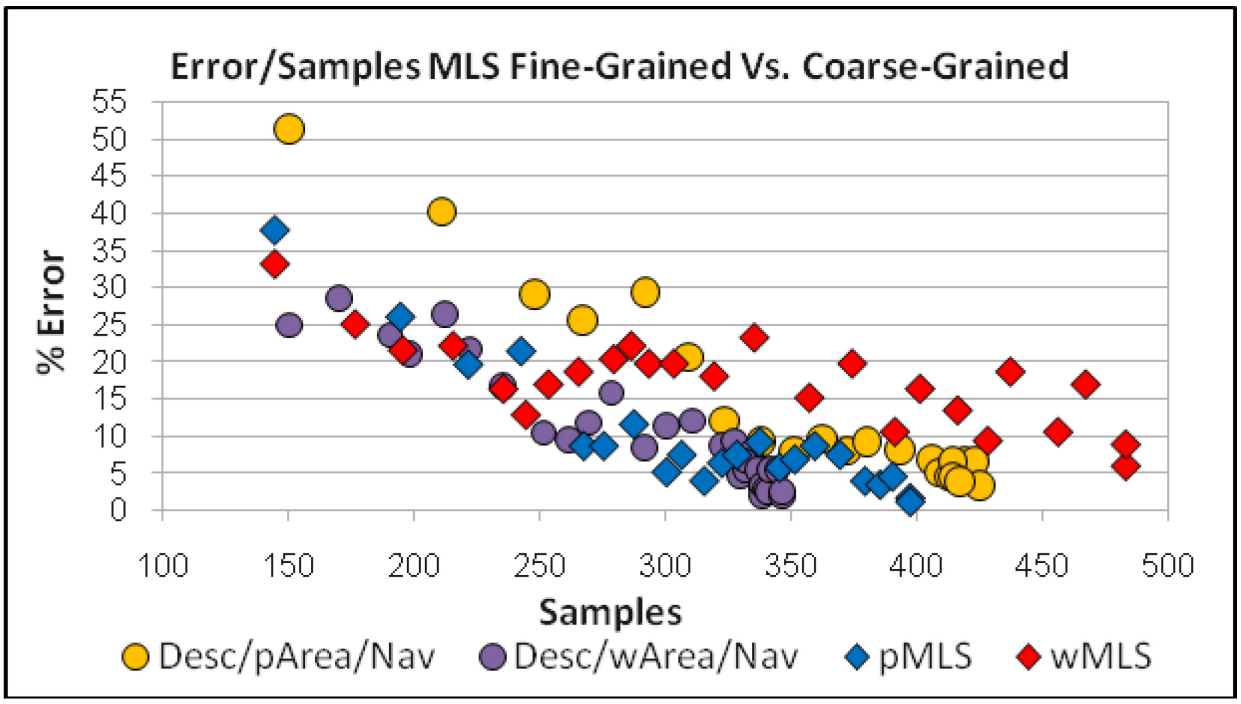

Figure 19: Error/Samples MLS Fine-Grained vs. Coarse-Grained

Figure 19 above compares two fine-grained partitions of MLS against the coarse-grained. The orange shows the results when training the Area sub-context with tri-phones (pArea), while the purple is when Area is trained with word models (wArea). Recall that Area is an awkward size of 88 vocabulary entries resulting in this confusion. Figure 19 shows that fine-grained with wArea converges about 50 samples earlier than pMLS. Yet pMLS achieves a better error rate of 1.7% compared to Desc/wArea/Nav, which reached only 2.46%. Fine-grained with pArea only exacerbated the situation, requiring more training than pMLS and achieving an error rate of only 3.98%.

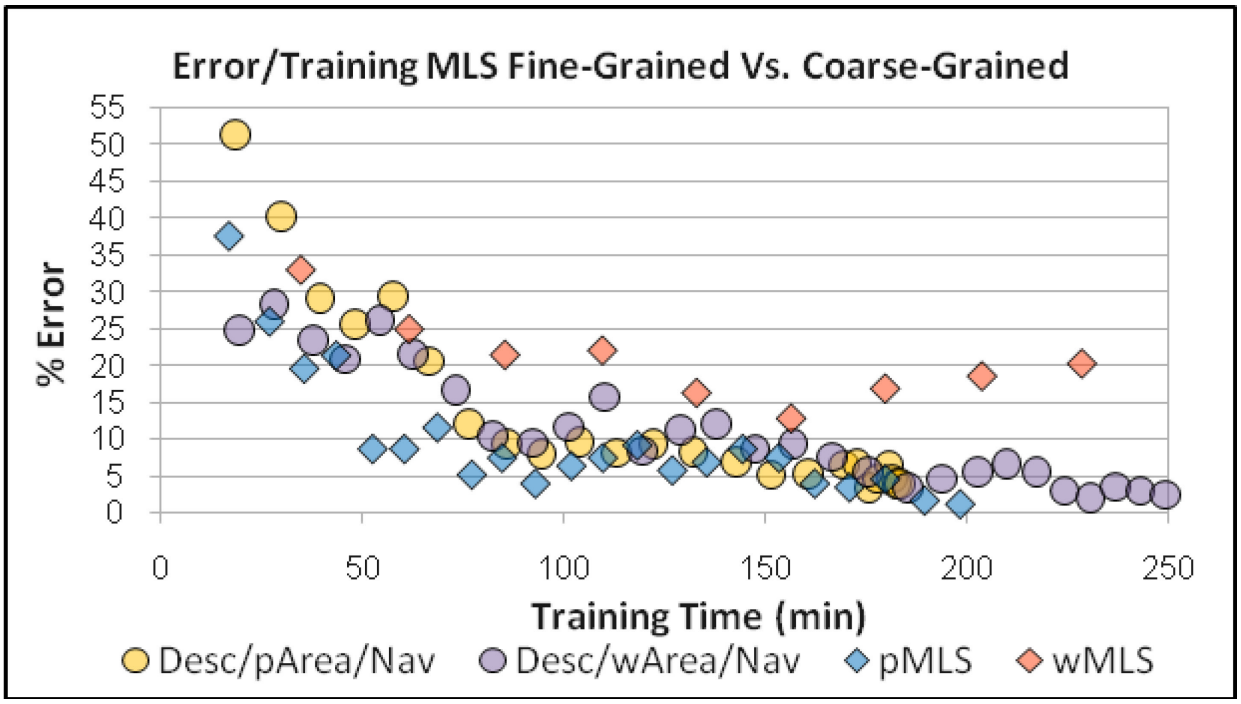

Figure 20: Error/Training MLS Fine-Grained Vs. Coarse-Grained

The training time plot in Figure 20 shows a slightly different trend than our sample plot did. Whereas fine-grained with wArea outperformed pArea in sample counts, the opposite is true with training time. In both plots, wMLS unsurprisingly performs poorly. Overall, the sub-contexts were not successful with MLS. This is obviously in sharp contrast to our results with MLB. This discrepancy merited further investigation.

6.2.3 MLB/MLS Comparison

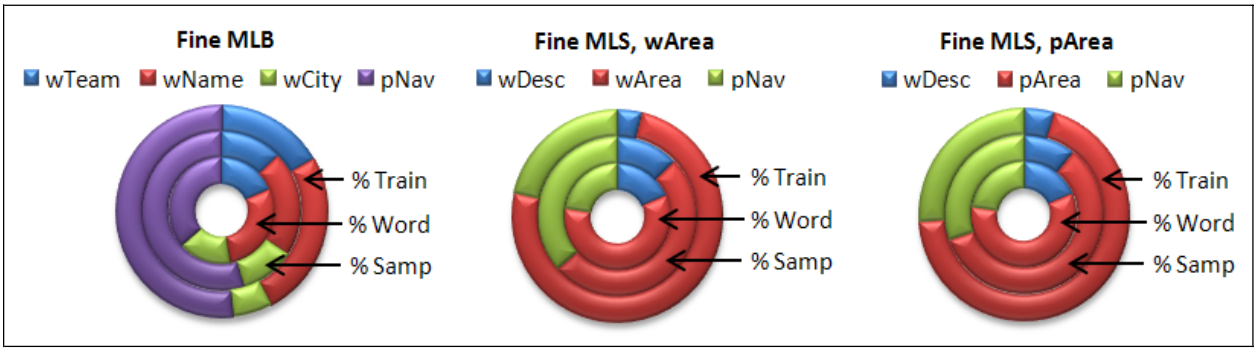

Figure 21: MLB/MLS Sub-Contextual Breakdown

To better understand the difference between fine-grained partitioning of MLB and MLS, we compared the contributions of each sub-context for each case. Specifically we compared the percentage of words of the entire context contained by each sub-context to the percentage of total

MLB Sub-Contexts

	Type	Words	% Words	Samples	% Samples	Training	% Training
Team:	W	30	18.40%	31	12.81%	8.51	16.80%
Name:	W	47	28.83%	52	21.49%	12.92	25.50%
City:	W	27	16.56%	27	11.16%	3.06	6.04%
Nav:	P	59	36.20%	132	54.55%	26.17	51.66%
Totals:	W	104	63.80%	110	45.45%	24.49	48.34%
	P	59	36.20%	132	54.55%	26.17	51.66%

Table 5: MLB Sub-Context Training/Sample Contributions

training and samples consumed by that sub-context, as shown in Figure 21 and the following tables. Looking at Table 5 we can note just how efficient word models (type W) were for MLB. Percentage wise, each sub-context uses less training and fewer samples proportional to their vocabulary size. Totaling the contributions of word models and tri-phone models separately, we can see that while the word models contain 63.8% of the MLB vocabulary, they consume only 45.45% and 48.34% of the samples and training respectively. The tri-phone trained Navigation sub-context meanwhile contains 36.20% of the vocabulary but consume 54.55% of the samples and 51.66% of the training time. It's clear that the word models are more efficient than the tri-phone models for MLB.

Taking a look at Table 6 below, with MLS containing wArea, it becomes clear how differently Area performs when trained with word models than other sub-contexts. While proportionally it uses slightly less samples than its vocabulary size of 58.67%, it consumes nearly three quarters of the total training time. This is a result of how SPHINX trains context-dependent acoustic models, which we use for all of our partitions. Recall that context-dependent acoustic models are generated by 3-tuple relations of a speech unit and its neighbors. When SPHINX creates context-dependent models it enumerates all possible combinations of units – not

MLS Sub-Contexts

	Type	Words	% Words	Samples	% Samples	Training	% Training
Desc:	W	28	18.67%	44	12.72%	6.13	3.78%
wArea:	W	88	58.67%	176	50.87%	121.17	74.78%
Nav:	P	34	22.67%	126	36.42%	34.73	21.43%
Totals:	W	116	77.33%	220	63.58%	127.3	78.57%
	P	34	22.67%	126	36.42%	34.73	21.43%

Table 6: MLS Sub-Context Training/Sample Contributions with wArea

just the combinations actually used in the vocabulary. Thus, increasing the number of speech units drastically increases the number of context-dependent units SPHINX must enumerate, lengthening the training process. Our Area partition contains 88 words, which correspond to 88 units when trained with word models – over double the number of units when compared to our tri-phone models, which are trained with a standard 40 phoneme set. This explains why Area, as well as wMLB and wMLS, performed poorly in regards of training time. It may be that for larger word trained vocabularies, leveraging context-independent acoustic models, though performing worse in accuracy initially, allows for faster training and better overall accuracy. Further work will have to be performed to determine if this is the case.

MLS Sub-Contexts

	Type	Words	% Words	Samples	% Samples	Training	% Training
Desc:	W	28	18.67%	44	10.55%	6.13	4.60%
pArea:	P	88	58.67%	247	59.23%	92.52	69.37%
Nav:	P	34	22.67%	126	30.22%	34.73	26.04%
Totals:	W	28	18.67%	44	10.55%	6.13	4.60%
	P	122	81.33%	373	89.45%	127.25	95.40%

Table 7: MLS Sub-Context Training/Sample Contributions with pArea

Unfortunately things do not improve much when training with pArea, as seen in Table 7. Although Area now consumes about half an hour less training time, it also uses about 70 more samples. Essentially, Area performs relatively poorly when trained with either word or tri-phone models. This is a function of its vocabulary size. It is both too large for word models, but not large enough to train well proportionally with tri-phone models. Tri-phones have the advantage (and disadvantage) of different words with similar sounds reinforcing one another, and from what we can tell, our 88 word vocabulary doesn't leverage this well when compared to pMLS.

6.3 Dynamic Corpus Comparison

Our final experiment compares the effects of changing the MLB player roster component of our vocabulary across the affected partitions. Examining the results in Table 8 below, we can quickly see that our word modeled Name sub-context performed more consistently in terms of accuracy than any of our phoneme partitions.

The change in errors was relatively inconsequential – actually improving accuracy in the removal test. Names did have one error that cropped up in the add players test, but we suspect this may have been an intermittent error present before the experiment. Essentially, a word trained model should experience no reduction in accuracy from removing vocabulary entries, because samples from each vocabulary entry only affect their related entry. With tri-phone models however, each sample models several tri-phones, thus removing a vocabulary entry and its samples will affect any word that shares any of the same tri-phones. Adding entries with word models should demonstrate similar consistency, incurring errors only if the word added is confusable with an entry already present in the vocabulary.

Names also trained faster than any of the other corpuses, both due to its size and its usage of word models. In the addition test, Names required 97.8% less training than Unified and 94.5% less training than the coarse-grained partition MLB+Gen. This is a benefit of fine-grained CP; by separating the dynamic parts of our vocabulary from the static, and using word models so that the effects of removals and additions would be more constrained, we are able to minimize the accuracy impact of changes while also minimizing retraining time. This requires several assumptions, mostly that the corpus is possible to be split in the first place, but also that the dynamic corpus not grow in net size, otherwise one would be forced to use tri-phone models. In

	Remove Lost Players		**Add New Players**		
Partition	Δ Errors	Training (min)	Δ Errors	Training (min)	Iterations
Name	-0.333	0.694	0.667	1.62	1
MLB	11	8.88	10	59.5	6
MLB+Gen	6.33	8.64	2.67	29.7	3
Unified	3	16.4	2	74.3	4

Table 8: Dynamic Corpus Experiment Results

situations where this is possible however, such as our Name sub-context, it is beneficial to do so.

For the tri-phone models a balance is struck. Larger models tend to be more robust against accuracy impairments due to changes. Larger models would have more samples, which would likely reinforce the tri-phone set of a corpus such that sample removal may not be as detrimental. This is supported by our results, showing a trend of accuracy improvement moving from MLB to MLB+Gen to Unified. Training time however is more interesting. The iterations column shows how many adjustments had to be made to the sample count before our 0% error condition was met. MLB, containing almost 30 words less than MLBGen, required considerably more iterations resulting in a greater net training time. Unified, meanwhile, used just slightly more iterations than MLB+Gen, but required considerably more training time. Thus, too small a vocabulary seems to lead to increased training time, but so does too large. A balance therefore must be struck between vocabulary size, robustness in accuracy, and retraining time. Although MLB+Gen, when compared to Unified, suffers from nearly twice the errors generated by removing vocabulary entries, its performance is nearly the same on the addition test. Furthermore, MLB+Gen for both tests consumes about half of the training time of Unified. Thus, the MLB+Gen coarse-grained partition still appears to be a decent choice, if a word-trained sub-context, such as Names, was not an option due to the limitations mentioned earlier.

6.4 Summary

All three of our experiments show positive results from contextual division of our vocabulary, in slightly varying manners. Whereas a unified vocabulary remained at about the same accuracy across increasing training, coarse-grained CP achieved higher accuracy as training increased, particularly when duplicating the general vocabulary into both contextual partitions. In this case, a GP was not beneficial. Yet, as mentioned before, a GP becomes more important when using sub-contexts, of which multiple will need to pull from that same general vocabulary. The fine-grained approach assumes a separate GP and shows that the results can be dramatically positive if a contextual partition can be divided into small sub-contexts (~50 entries or less) predominately trained with word HMMs. Fine-grained partitioning of MLB improved accuracy by 92% and 88% over coarse-grained MLB when compared at similar sample and training respectively. If, however, a contextual partition cannot be neatly divided, the benefits fall off

sharply, and it instead better to keep the contextual vocabulary as a whole single context, rather than breaking into sub-contexts. This was determined by our results from attempting to divide MLS and finding that over half of our vocabulary fell into our Area sub-context. If Area could have been divided neatly into smaller sub-contexts, we see no reason why MLS would not have performed just as well as MLB. Area's vocabulary size simply fell into a range that neither worked well with word models or tri-phone models. It could be that there is another model type that would solve this problem, but that would be an exploration for later work. The Dynamic Corpus experiment provides yet another reason to divide a unified case. By separating our dynamic element into a word trained sub-context, retraining with new vocabulary became trivial.

Given all this information we are able to make a final decision regarding how to process our vocabulary. Sub-contextual division of MLB is definite, and the dynamic aspect of our Names sub-context performs much better when separated. In using our sub-contexts we also leverage a GP. That leaves MLS, which neither divides well, nor has any aspect to its vocabulary that is especially dynamic. The introduction of a new city or county could change our vocabulary, but this would happen relatively infrequently compared to MLB roster changes. Due to this, keeping MLS as a whole context is the most effective. By combining the results from all of these partitions, and comparing to the performance of our unified baseline, we obtain a final set of plots.

One should note that this is the same unified that was trained for coarse-grained using 2 samples per entry as a base. While one may think this puts Unified at a samples disadvantage, since our MLB sub-contexts were trained starting with 1 sample, we attempted to train Unified starting with 1 sample per entry, and it performed even worse. For terminology, we represent that MLB is trained with sub-contexts in the plot by labeling it as 'mlb' in lowercase.

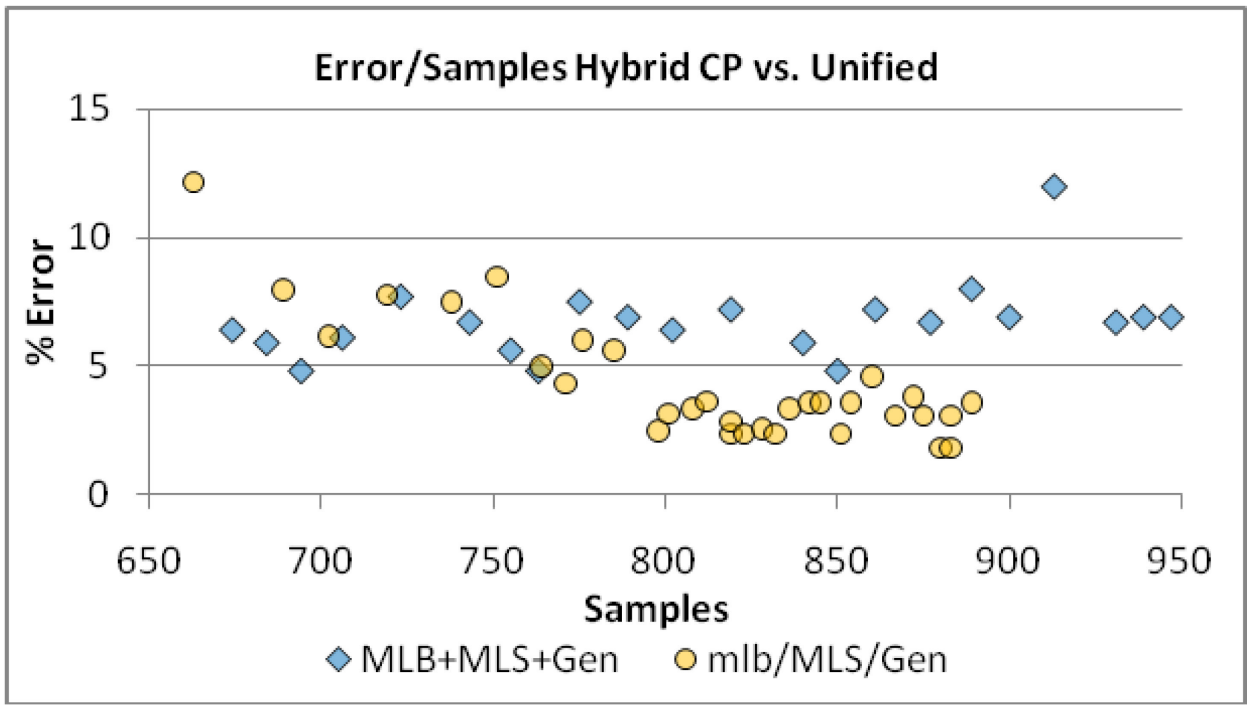

Figure 22: Error/Samples Hybrid CP vs. Unified

Figure 22 provides a comparison of our hybrid CP against the baseline in terms of accuracy and sample range. Our sub-contexts/context hybrid converges rapidly and achieves a best error rate of 1.87%, while Unified's best error rate was 4.8%. This represents a 61% reduction. The fact that Unified doesn't improve with added samples suggests that it will not converge to the error rate found by our optimized model without significant additional training. Similary, the hybrid CP approach is efficient with training, quickly overtaking the baseline in the accuracy/training plot of Figure 23.

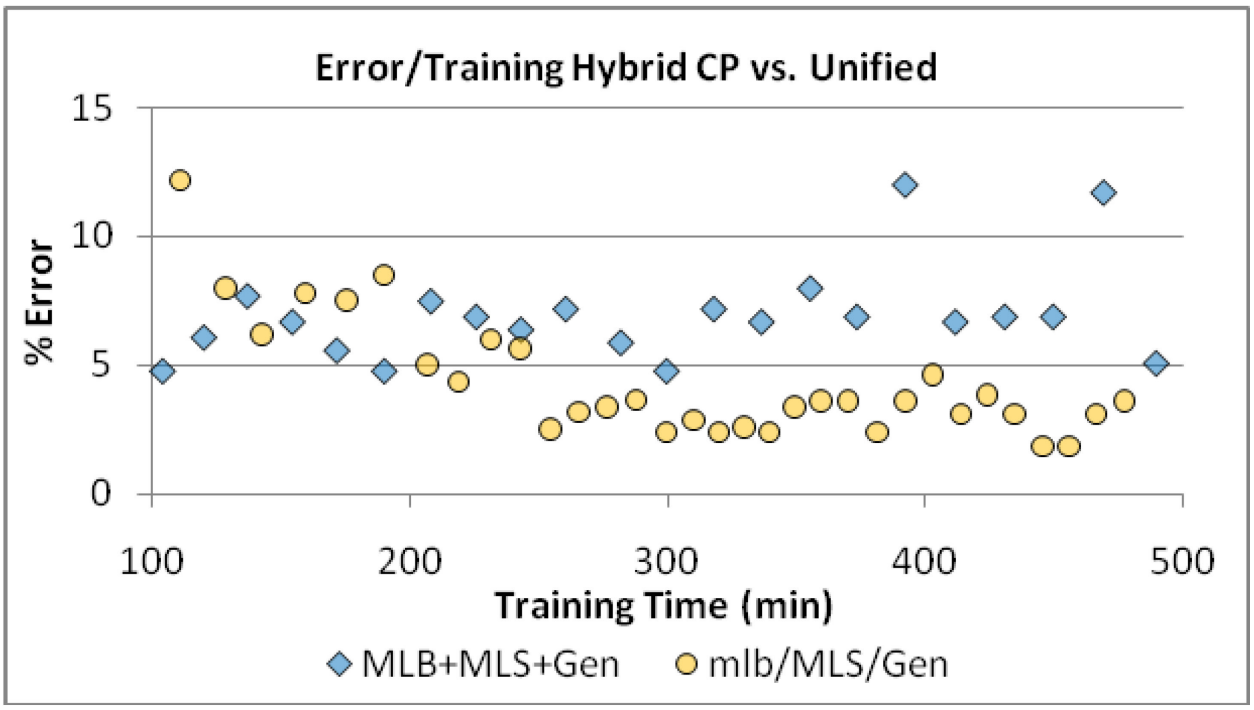

Figure 23: Error/Training Hybrid CP vs. Unified

Thus far, our contextual partitioning approach has demonstrated the following qualities:

1) Efficiency in training, achieving higher accuracy at similar sample/training ranges:
    a. Coarse-grained partitioning improves accuracy up to 51.9% over unified
    b. Fine-grained partitioning improves accuracy up to 92% over coarse-grained – so long as the coarse-grained partition can be cleanly divided into sub-contexts
    c. Hybrid coarse/fine grained partitioning improves accuracy up to 61% over unified
2) Robustness to vocabulary changes through fine-grained separation of the dynamic aspect of our vocabulary
    a. Fine-grained CP experienced essentially no additional errors from removing or adding our dynamic vocabulary entries, while coarse-grained and unified approaches ranged from 2 – 11 additional errors.
    b. Fine-grained CP required up to 97.8% less training to recognize new vocabulary words than other approaches.

CP, therefore, has demonstrated dominance over a unified approach in two of our three performance metrics: error/training efficiency, and robustness. Our, final performance metric,

decoding speedup remains. We discuss this last variable for CP in terms of scalability over increasing core counts, and in comparison to a TLP approach.

7 Scalability

Previous research has shown that decoding time is related linearly to vocabulary size[11]. TLP speedup for speech recognition does not scale linearly with increasing core counts however, limiting vocabulary expansion.

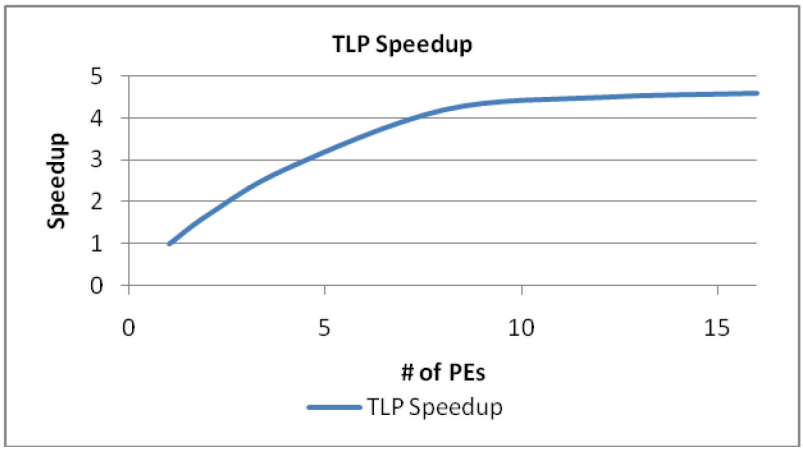

Figure 24: TLP Speedup for SR from [38].
Illustrated under "Fair Use" copyright guidelines.

Consider a vocabulary containing M potential contextual partitions, the size of each represented by $|C_m|$, for processing on a chip of M cores. TLP ignores the M contexts, and attempts to leverage task/thread level parallelism across the entire vocabulary. Thus the decoding time for TLP is represented by Equation 1, where X is the decoding time for the vocabulary on a uni-processor.

$$TLP_M\ Decoding\ Time = \left(\sum_{m=1}^{M} |C_m|\right) * \frac{X}{TLP_M\ Speedup}$$

Equation 1: TLP Decoding Time for M Cores/Contexts

Then consider CLP, where the vocabulary is divided into those M contexts for processing by separate PEs. Assuming that for M cores, one can obtain M contexts, and the vocabulary is evenly divided by those M contextual partitions, decoding time instead becomes a function of the number of partition/core assignments.

$$Ideal\ CLP_M\ Decoding\ Time = \left(\sum_{m=1}^{M} |C_m|\right) * \frac{X}{M}$$

Equation 2: Ideal CLP Decoding Time for M Cores/Contexts

Thus, each processor is assigned 1/M percent of the entire vocabulary. As M increases, the vocabulary assigned to each processor shrinks (again assuming contexts can continue to be found, and the vocabulary is divided evenly). Since X, the uni-processor decoding time, is linearly related to vocabulary size, linear shrinkage of vocabulary results in a linear decrease in decoding time; therefore, in this ideal sense, CLP provides linear speedup across increasing cores.

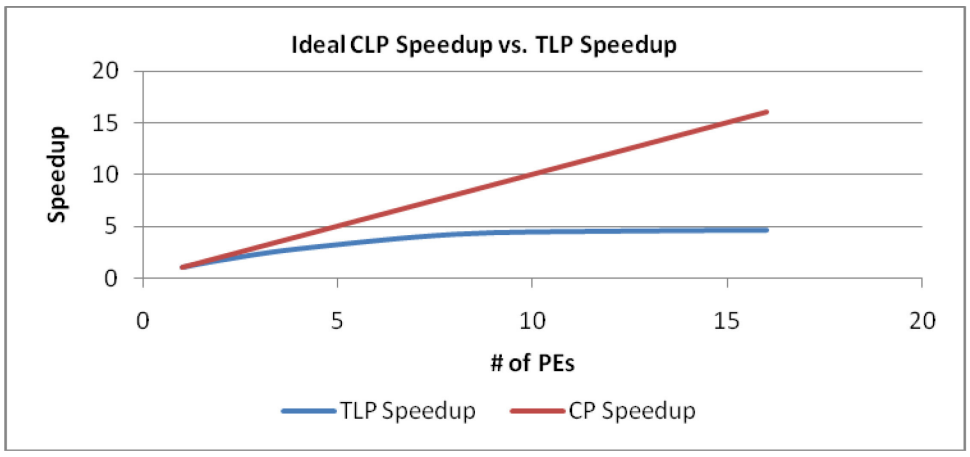

Figure 25: Ideal CLP Speedup vs. TLP Speedup from [38].
TLP Speedup illustrated under "Fair Use" copyright guidelines.

Clearly there is a limit on the number of times a vocabilary can be sub-divided. However, for many applications, including a user-interface, one only needs to reduce decoding time to allow for real-time recognition. Assuming a vocabulary with M contextual partitions and M cores achieves real-time recognition, additional cores could be used to add new partitions, and thus new vocabulary. So long as the M+1th, and so on, partition(s) were no larger than any of the existing partitions that were decodable in real-time, average decoding speed would not suffer. Our assumption earlier that the vocabulary was evenly divided by the M contexts was for simplicity, but in practicality the only requirement is that the largest context be decodable in real-time for a user interface. If this is true, then any of the smaller contexts will also be decodable in real-time (assuming symmetric processing elements). Thus, while a fixed size vocabulary would be unlikely to allow for linear speedup as shown by our ideal CLP case, if given a vocabulary

processable in real-time, it is quite possible one could achieve linear increase in vocabulary, and thus linear speedup when compared to processing the increasing vocabulary with a uni-processor.

Another consideration that may factor into decoding speedup is the ability to cache small partitioned acoustic models. If contextually partitioned acoustic models require less space than a unified approach, it may be possible to cache active models leading to decoding speedup.

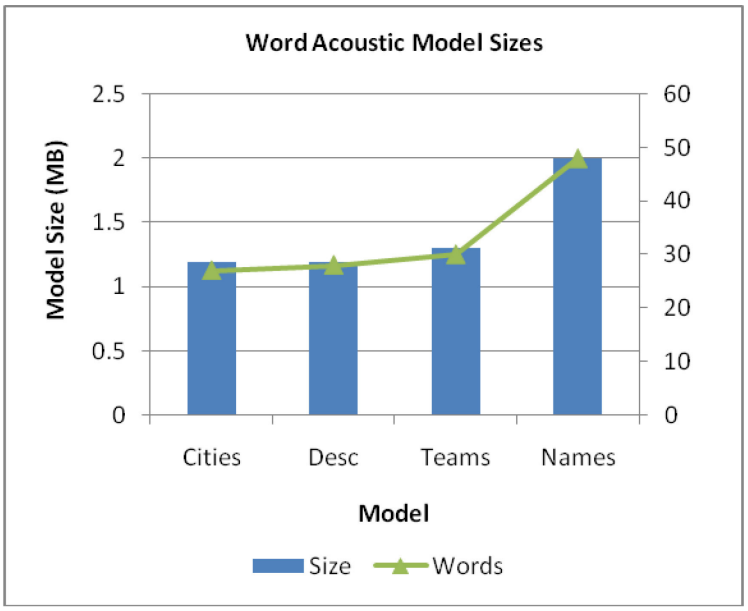

Figure 26: Word Acoustic Model Sizes

Figure 26 above shows the sizes of our word trained acoustic models. Since our word trained models are using untied context-dependent models, their size is dependent predominantly on the number of words in the vocabulary. Since we keep these vocabularies small, the model sizes are small – around 2 megabytes or less. Figure 27 below shows the sizes of our tri-phone trained acoustic models. For smaller vocabularies, these depend on number of words and speech units, but once our vocabulary reaches the size of MLB Nav, at 59 words, instead the primary factor in model size is the number of senones. Since we kept the number of senones constant at 1000 for all our models, our larger tri-phone acoustic models hovered around the same size, between 6 and 7 megabytes. The number of senones one can use is dependent on the amount of training available[102], thus it seems likely that larger vocabularies will use greater numbers of senones, since larger vocabularies tend to require, and thus have, more training. Larger vocabularies will therefore tend to correspond with larger acoustic models. Although summing the size of our

partitions leads to greater overall storage space requirements, the possibility of caching only the acoustic model necessary for a given task is enticing.

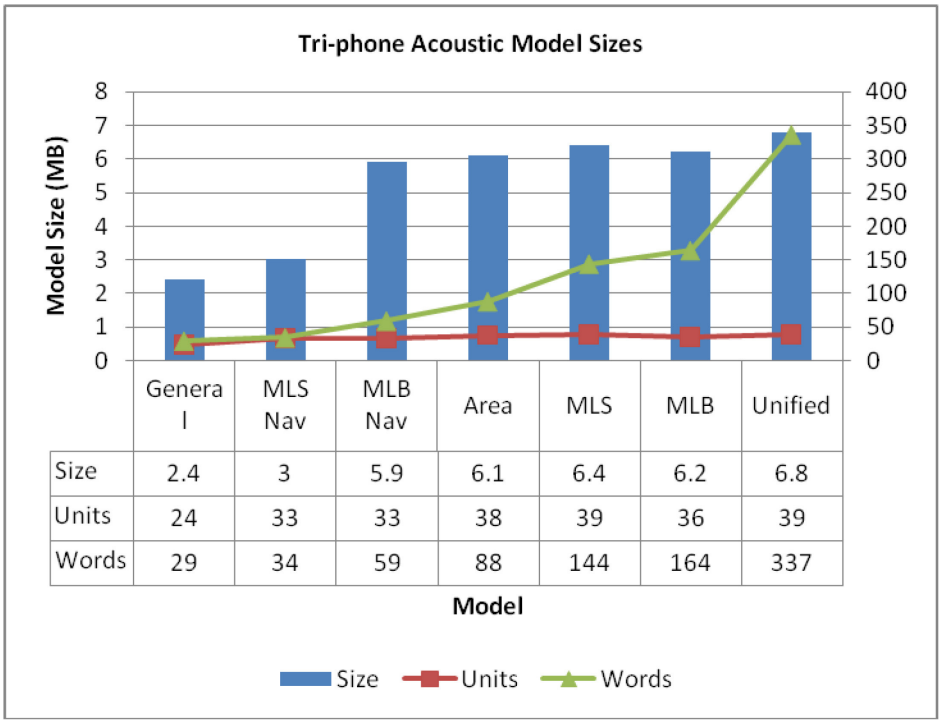

Figure 27: Tri-phone Acoustic Model Sizes

A final consideration for scalability is power. Our architecture aims to break a vocabulary into its simplest contexts for processing, using as many context/core assignments as possible. Previous work has shown that leveraging many cores at a lower clock speed provides power efficiencies compared to fewer larger cores at higher clocks [103, 104]. Our architecture would enable better utilization of higher core counts, while also taking advantage of the power efficiencies of such chip designs. ASMPs are known to provide the best power efficiency of chip designs, through assignment of cores based on processing need. Similarly, our architecture can assign cores to contexts based on vocabulary size, such that larger vocabularies obtain the processing they need to maintain real-time decoding, without over-provisioning for small vocabularies. It may also be possible to deactivate context/core assignments that are deemed inappropriate for the current situation, increasing power efficiency further.

Two additional overheads have thus far been ignored by our calculations. First we have overhead associated with identifying contexts. In order to leverage M cores, we have to have M contexts to assign. Ideally the device would perform context identification as a result of normal use, since there is a significant amount of information about contextual use already available in

web browsers. This process will surely have computational requirements of its own, reducing our ideal speedup. But, we believe that context identification will only impede the speedup slightly. Coarse-grained contextual partitions should be identifiable by usage trends. The latest report from the U.S. Census Bureau on Computer and Internet Use in the United States[105] shows that the types of tasks one uses computers for at the office and at home are different. Furthermore, surveys of broadband activity[106] suggest that residential and workplace usage patterns differ in session duration, bytes transferred, number of sessions, time of access, and service type. Information such as this can be used to define categories of usage, leading to partitions. For fine-grained partitioning, the device can examine usage within these broad classifications; monitoring details such as input field order for different search forms could lead to partitions similar to our Name, Team, or City sub-contexts.

A second overhead to consider is the context activation algorithm, and output selection. Determining which context to activate in the recognition process will also come at a cost. If multiple contexts are active, output selection will have to occur delaying decoding time. Furthermore, if the wrong context is selected, either by the CAA or at output selection, accuracy will suffer as well. This situation will become more difficult as the number of contexts increases. How a context activation algorithm should cope with this increased load is an interesting question. One potential answer might be to assign activation probabilities not only on the situation of the user, perhaps based on current environment as discussed earlier, but also based on frequency of use. Our human memory has difficult recalling things used less frequently or less recently. We might have trouble placing something we haven't seen in a while into its proper context, yet often enough we accomplish this at our best-effort. Similarly the algorithm itself would converge to a best-effort computing model where it might take a series of tries to recall something from an old context. These are all suppositions for further research however. The goal of our research, to show that contextual partitioning can be effective for speech recognition systems, was successful, and that in itself should be foundation enough to explore these other issues in more depth.

8 Conclusion and Future Work

This research contributes an exploration of a contextually partitioned architecture, setting the precursor for a new level of parallelism. Targeting speech recognition as our application, in contrast to a unified approach, CP demonstrates:

1) Efficiency in training, achieving up to 61% higher accuracy
2) Robustness to vocabulary changes, incurring essentially no additional errors from word replacement, and requiring up to 97.8% less training for new vocabulary words
3) Potential to scale nearly linearly with increased chip core counts, allowing for consistent increases in vocabulary, while maintaining real-time recognition
4) Potential to cache specific acoustic models for further reduction of decoding time

First, improved training efficiency is useful for speaker-dependent, and potentially speaker-adaptive approaches, improving user acceptance of a speech driven interface. Second, fine-grained partitioning allows for usage of acoustic models trained with different units specific to the size of the partition. Word unit usage for small vocabulary sets allows for greater training efficiency, high accuracy, and robustness against changes not possible with large tri-phone acoustic models. Third, CP's potential to scale well with increasing core counts presents an architecture capable of utilizing the core counts expected of future chip designs, a trait not shared by TLP. This scalability is important, in regards to speech recognition, because it allows for more complex speech user interfaces, a challenging but desirable market.

Our results provide a reasonable basis for performing further research on contextual partitioning. Many unanswered questions remain, particularly leading to our overall vision of a MISD architecture. First, simulation and/or deployment on embedded systems will help to determine scalability in real-world practice. This will entail research evaluating different context activation algorithms and output selection schemes in terms of accuracy and overhead. Further explorations will have to evaluate specific context definition methods, operating at run-time, based on overhead and ability to generate enough contexts to fill available cores effectively. Additional speech units, other than tri-phones and word, may also be evaluated to obtain a full

continuum of vocabulary size / speech unit mappings. In addition, research will have to compare different, potentially dynamic, grammar model mechanisms including inter-partition relationships. Culmination of such work with ours will result in a holistic view of CP when applied to single-input single-output situations. In order to fully realize our MISD architecture vision, and leverage techniques similar to the brain, further questions remain to be answered. Primarily, research into how to effectively use multiple outputs, as opposed to simply selecting one of a set, must be pursued. Answering this question allows for architectures that employ parallel partitions operating on the same data much as the brain is thought to. Understanding the implications of this brain-inspired approach to computer architectures may ultimately transform how parallelism is viewed in computing, ushering in entirely new and exciting avenues of research.

References

[1] B. Crothers, "Intel says to prepare for 'thousands of cores'," *Nanotech: The Circuits Blog*, Oct. 9th, CNET, 2008.

[2] J. L. Hennessy, and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*: Morgan Kaufmann Publishers Inc., 2006.

[3] M. S. Gazzaniga, and E. Bizzi, "The Cognitive Neuroscience of Categorization," *The New Cognitive Neurosciences*: MIT Press, 2000.

[4] B. Dubuc, "The Brain From Top To Botom," McGill University, 2002.

[5] M. J. M. Steven Brown, Lawrence M. Parsons,, "Music and language side by side in the brain: a PET study of the generation of melodies and sentences," *European Journal of Neuroscience,* vol. 23, no. 10, pp. 2791-2803, 2006.

[6] M. Herlihy, and V. Luchangco, "Distributed computing and the multicore revolution," *SIGACT News,* vol. 39, no. 1, pp. 62-72, 2008.

[7] M. Somers, and J. M. Paul, "Webpage-based benchmarks for mobile device design," in Proceedings of the 2008 Asia and South Pacific Design Automation Conference, Seoul, Korea, 2008.

[8] E. Finegan, and J. R. Rickford, *Language in the U.S.A.: Themes for the 21st century*, New York: Cambridge University Press, 2004.

[9] M. E. Wheeler, G. L. Shulman, R. L. Buckner *et al.*, "Evidence for Separate Perceptual Reactivation and Search Processes during Remembering," *Cereb. Cortex,* vol. 16, no. 7, pp. 949-959, July 1, 2006, 2006.

[10] L. D. Paulson, "Speech Recognition Moves from Software to Hardware," *Computer,* vol. 39, no. 11, pp. 15-18, 2006.

[11] J. Suontausta, J. Hakkinen, and O. Viikki, "Fast decoding in large vocabulary name dialing." pp. 1535-1538 vol.3.

[12] S. Yu, and C. Eric, "Studies in massively speaker-specific speech recognition." pp. I-825-8 vol.1.

[13] J. Larus, "Spending Moore's dividend," *Commun. ACM,* vol. 52, no. 5, pp. 62-69, 2009.

[14] V. Pankratius, C. Schaefer, A. Jannesari *et al.*, "Software engineering for multicore systems: an experience report," in Proceedings of the 1st international workshop on Multicore software engineering, Leipzig, Germany, 2008.

[15] L. H. Leong, S. Kobayashi, N. Koshizuka *et al.*, "CASIS: a context-aware speech interface system," in Proceedings of the 10th international conference on Intelligent user interfaces, San Diego, California, USA, 2005.

[16] M. Coen, L. Weisman, K. Thomas *et al.*, "A Context Sensitive Natural Language Modality for an Intelligent Room," *Proceedings of MANSE'99*, 1999.

[17] K. M. Everitt, S. Harada, J. Bilmes *et al.*, "Disambiguating speech commands using physical context," in Proceedings of the 9th international conference on Multimodal interfaces, Nagoya, Aichi, Japan, 2007.

[18] D. Siewiorek, A. Smailagic, J. Furukawa *et al.*, "SenSay: a context-aware mobile phone." pp. 248-249.

[19] J. Ho, and S. S. Intille, “Using context-aware computing to reduce the perceived burden of interruptions from mobile devices,” in Proceedings of the SIGCHI conference on Human factors in computing systems, Portland, Oregon, USA, 2005.
[20] L. Cristoforetti, M. Matassoni, M. Omologo *et al.*, "Use of parallel recognizers for robust in-car speech interaction." pp. I-320-I-323 vol.1.
[21] M. J. Flynn, and K. W. Rudd, “Parallel architectures,” *ACM Comput. Surv.,* vol. 28, no. 1, pp. 67-70, 1996.
[22] J. M. Paul, M. Otoom, M. Somers *et al.*, "The Emerging Landscape of Computer Performance Evaluation," *Advances in Computers*, M. V. Zelkowitz, ed., pp. 235-280, Burlington: Academic Press, 2009.
[23] S. H. Roosta, *Parallel Processing and Parallel Algorithms: Theory and Computation*: Springer, 1999.
[24] M. Neschen, "COLUMNUS-an architecture for multi-spin-coding algorithms." pp. 139-150 vol.1.
[25] A. Halaas, B. Svingen, M. Nedland *et al.*, “A recursive MISD architecture for pattern matching,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 12, no. 7, pp. 727-734, 2004.
[26] R. Singh. "The Sphinx Speech Recognition Systems," http://www.cs.cmu.edu/~rsingh/homepage/sphinx_history.html.
[27] E. Alpaydin, and F. Gürgen, "Comparison of Statistical and Neural Classifiers and Their Applcations to Optical Character Recognition and Speech Classification," *Neural network systems techniques and applicaions*, pp. 61-88: Academic Press, 1996.
[28] J. P.-S. R. Solera-Ureña, D. Martín-Iglesias, A. Gallardo-Antolín, C. Peláez-Moreno, F. Díaz-de-María, "SVMs for Automatic Speech Recognition: A Survey," *Progress in Nonlinear Speech Processing*, Lecture Notes in Computer Science, pp. 190-216: Springer Berlin / Heidelberg, 2007.
[29] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proceedings of the IEEE,* vol. 77, no. 2, pp. 257-286, 1989.
[30] N. D. Warakagoda. "A Hybrid ANN-HMM ASR system with NN based adaptive preprocessing," http://jedlik.phy.bme.hu/~gerjanos/HMM/hoved.html.
[31] “Understanding speech: an interview with John Makhoul,” *Signal Processing Magazine, IEEE,* vol. 22, no. 3, pp. 76-79, 2005.
[32] G. A. Fink, "Configuration of Hidden Markov Models," *Markov Models for Pattern Recognition*, pp. 127-136: Springer Berlin Heidelberg, 2008.
[33] D. A. Reynolds, “Large population speaker identification using clean and telephone speech,” *Signal Processing Letters, IEEE,* vol. 2, no. 3, pp. 46-48, 1995.
[34] A. I. Rudnicky. "Hub 4: Business Broadcast News," Oct. 28th, 2009; http://www.speech.cs.cmu.edu/air/papers/hub4/proceedings_paper1.html.
[35] X. Aubert, C. Dugast, H. Ney *et al.*, "Large vocabulary continuous speech recognition of Wall Street Journal data." pp. II/129-II/132 vol.2.
[36] X. Huang, and K. F. Lee, “On speaker-independent, speaker-dependent, and speaker-adaptive speech recognition,” *Speech and Audio Processing, IEEE Transactions on,* vol. 1, no. 2, pp. 150-157, Apr, 1993.
[37] B. Raj, and E. W. D. Whittaker, "Lossless compression of language model structure and word identifiers," *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing (Cat. No.03CH37404)*. pp. 388-91.

[38] S. Phillips, and A. Rogers, “Parallel Speech Recognition,” *International Journal of Parallel Programming,* vol. 27, no. 4, pp. 257-288, 1999.
[39] "About function MRI(General)," May 24, 2009, 2009; http://www.fmri.org/fmri.htm.
[40] B. J. Shannon, and R. L. Buckner, “Functional-Anatomic Correlates of Memory Retrieval That Suggest Nontraditional Processing Roles for Multiple Distinct Regions within Posterior Parietal Cortex,” *J. Neurosci.,* vol. 24, no. 45, pp. 10084-10092, November 10, 2004, 2004.
[41] M. E. Wheeler, and R. L. Buckner, “Functional Dissociation among Components of Remembering: Control, Perceived Oldness, and Content,” *J. Neurosci.,* vol. 23, no. 9, pp. 3869-3880, May 1, 2003, 2003.
[42] L. M. Parsons, and D. Osherson, “New Evidence for Distinct Right and Left Brain Systems for Deductive versus Probabilistic Reasoning,” *Cereb. Cortex,* vol. 11, no. 10, pp. 954-965, October 1, 2001, 2001.
[43] S. Glazewski, B. L. Benedetti, and A. L. Barth, “Ipsilateral Whiskers Suppress Experience-Dependent Plasticity in the Barrel Cortex,” *J. Neurosci.,* vol. 27, no. 14, pp. 3910-3920, April 4, 2007, 2007.
[44] S. Begley, "How The Brain Rewires Itself," *TIME*, http://www.time.com/time/magazine/article/0,9171,1580438-1,00.html, [May 30th, 2009, 2007].
[45] D. Poeppel, W. J. Idsardi, and V. van Wassenhove, “Speech perception at the interface of neurobiology and linguistics,” *Philosophical Transactions of the Royal Society B: Biological Sciences,* vol. 363, no. 1493, pp. 1071-1086, March 12, 2008, 2008.
[46] A. Rakotonirainy, "Trends and future of mobile computing." pp. 136-140.
[47] A. G. Hauptmann, and A. I. Rudnicky, "A comparison of speech and typed input," *HLT '90: Proceedings of the workshop on Speech and Natural Language*, Association for Computational Linguistics, 1990, pp. 219-224.
[48] P. Kocher, R. Lee, G. McGraw *et al.*, "Security as a new dimension in embedded system design," *Proceedings 2004. Design Automation Conference (IEEE Cat. No.04CH37531)*. pp. 753-60.
[49] R. Kumar, D. M. Tullsen, N. P. Jouppi *et al.*, “Heterogeneous chip multiprocessors,” *Computer,* vol. 38, no. 11, pp. 32-8, 2005.
[50] M. A. Suleman, O. Mutlu, M. K. Qureshi *et al.*, “Accelerating critical section execution with asymmetric multi-core architectures,” in Proceeding of the 14th international conference on Architectural support for programming languages and operating systems, Washington, DC, USA, 2009.
[51] T. Y. Morad, U. C. Weiser, A. Kolodnyt *et al.*, “Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors,” *Computer Architecture Letters,* vol. 5, no. 1, pp. 14-17, 2006.
[52] M. Becchi, and P. Crowley, “Dynamic thread assignment on heterogeneous multiprocessor architectures,” in Proceedings of the 3rd conference on Computing frontiers, Ischia, Italy, 2006.
[53] S. Fei, S. Ravi, A. Raghunathan *et al.*, “Application-specific heterogeneous multiprocessor synthesis using extensible processors,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 25, no. 9, pp. 1589-1602, 2006.

[54] D. P. Gulati, K. Changkyu, S. Sethumadhavan *et al.*, “Multitasking workload scheduling on flexible core chip multiprocessors,” *Computer Architecture News,* vol. 36, no. 2, pp. 46-55, 2008.
[55] I. Engin, M. Kirman, N. Kirman *et al.*, “Core fusion: accommodating software diversity in chip multiprocessors,” *Computer Architecture News,* vol. 35, no. 2, pp. 186-97, 2007.
[56] O. Moreira, F. Valente, and M. Bekooij, “Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor,” in Proceedings of the 7th ACM \& IEEE international conference on Embedded software, Salzburg, Austria, 2007.
[57] A. K. Coskun, T. S. Rosing, K. A. Whisnant *et al.*, "Temperature-aware MPSoC scheduling for reducing hot spots and gradients," *13th Asia and South Pacific Design Automation Conference ASP-DAC 2008.* pp. 49-54.
[58] W. Feng, C. Nicopoulos, W. Xiaoxia *et al.*, "Variation-aware task allocation and scheduling for MPSoC," *2007 IEEE/ACM International Conference on Computer Aided Design.* pp. 598-603.
[59] A. Andrei, P. Eles, P. Zebo *et al.*, "Predictable implementation of real-time applications on multiprocessor systems-on-chip," *2008 21st International Conference on VLSI Design.* pp. 103-10.
[60] S. Schliecker, M. Ivers, and R. Ernst, "Integrated analysis of communicating tasks in MPSoCs," *2006 4th IEEE/ACM/IFIPHardware/Software Codesign and System Synthesis (CODES+ISSS).* pp. 288-93.
[61] P. Partha Pratim, C. Grecu, M. Jones *et al.*, “Performance evaluation and design trade-offs for network-on-chip interconnect architectures,” *Computers, IEEE Transactions on,* vol. 54, no. 8, pp. 1025-1040, 2005.
[62] R. Beraha, I. Walter, I. Cidon *et al.*, "The design of a latency constrained, power optimized NoC for a 4G SoC." pp. 86-86.
[63] J. H. Bahn, S. E. Lee, and N. Bagherzadeh, "On Design and Analysis of a Feasible Network-on-Chip (NoC) Architecture." pp. 1033-1038.
[64] A. Raina, and V. Muthukumar, "Traffic Aware Scheduling Algorithm for Network on Chip." pp. 877-882.
[65] S. Mahadevan, M. Storgaard, J. Madsen *et al.*, "ARTS: a system-level framework for modeling MPSoC components and analysis of their causality," *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems.* pp. 480-3.
[66] G. Palermo, C. Silvano, and V. Zaccaria, "An efficient design space exploration methodology for on-chip multiprocessors subject to application-specific constraints," *2008 Symposium on Application Specific Processors (SASP).* pp. 75-82.
[67] K. Virk, and J. Madsen, "A system-level multiprocessor system-on-chip modeling framework," *2004 International Symposium on System-on-Chip (IEEE Cat. No.04EX876).* pp. 81-4.
[68] J. Palmer, and B. Nelson, "A Parallel FFT Architecture for FPGAs," *Field Programmable Logic and Application*, pp. 948-953, 2004.
[69] H. Suzuki, Y. Takagi, R. Yamaguchi *et al.*, "FPGA implementation of FDTD algorithm." p. 4 pp.
[70] A. Madanayake, L. Bruton, and C. Comis, "FPGA architectures for real-time 2D/3D FIR/IIR plane wave filters." pp. III-613-16 Vol.3.

[71] M. Shahzad, and S. Zahid, "Image coprocessor: a real-time approach towards object tracking," *2009 International Conference on Digital Image Processing, ICDIP*. pp. 220-4.
[72] V. Hampel, P. Sobe, and E. Maehle, “Experiences with a FPGA-based Reed/Solomon-encoding coprocessor,” *Microprocessors and Microsystems,* vol. 32, no. 5-6, pp. 313-20, 2008.
[73] N. Vassiliadis, G. Theodoridis, and S. Nikolaidis, “The ARISE approach for extending embedded processors with arbitrary hardware accelerators,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 17, no. 2, pp. 221-33, 2009.
[74] H. Chen, and F. Vahid, "Transmuting coprocessors: dynamic loading of FPGA coprocessors," *2009 46th ACM/IEEE Design Automation Conference (DAC)*. pp. 848-51.
[75] E. J. McDonald, "Runtime FPGA partial reconfiguration," *IEEE Aerospace Conference Proceedings*.
[76] L. Singhal, and E. Bozorgzadeh, “Multi-layer floorplanning for reconfigurable designs,” *IET Computers & Digital Techniques,* vol. 1, no. 4, pp. 276-94, 2007.
[77] J. Angermeier, and J. Teich, "Heuristics for scheduling reconfigurable devices with consideration of reconfiguration overheads," *Proceedings of the 2008 IEEE International Parallel Distributed Processing Symposium.* pp. 3596-603.
[78] E. El-Araby, M. Taher, M. Abouellail *et al.*, "Comparative Analysis of High Level Programming for Reconfigurable Computers: Methodology and Empirical Study." pp. 99-106.
[79] Q. Fengbin, Z. Xianyi, W. Shanshan *et al.*, "RCC: a new programming language for reconfigurable computing," *2009 11th IEEE International Conference on High Performance Computing and Communications (HPCC)*. pp. 688-93.
[80] G. V. Leming, and K. Nepal, "Low-power FPGA routing switches using adaptive body biasing technique," *2009 52nd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*. pp. 447-50.
[81] J. H. Anderson, and F. N. Najm, “Low-power programmable FPGA routing circuitry,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 17, no. 8, pp. 1048-60, 2009.
[82] A. Schmitt, D. Zaykovskiy, and W. Minker, "Speech recognition for mobile devices," *International Journal of Speech Technology*, Springer Science & Business Media B.V., 2008, pp. 63-72.
[83] R. Flynn, and E. Jones, “Robust distributed speech recognition using speech enhancement,” *Consumer Electronics, IEEE Transactions on,* vol. 54, no. 3, pp. 1267-1273, 2008.
[84] D. Chandra, U. Pazhayaveetil, and P. D. Franzon, "Architecture for Low Power Large Vocabulary Speech Recognition." pp. 25-28.
[85] J. W. Schuster, K. Gupta, and R. Hoare, "Speech silicon AM: an FPGA-based acoustic modeling pipeline for hidden Markov model based speech recognition." p. 4 pp.
[86] S. Nedevschi, R. K. Patra, and E. A. Brewer, "Hardware speech recognition for user interfaces in low cost, low power devices." pp. 684-689.
[87] B. X. Chen. "AT&T Developing Speech Recognition Potentially for iPhone," 4/28/2009, 2009; http://www.wired.com/gadgetlab/2008/07/att-developing.
[88] "Now you can speak to Google Mobile App on your iPhone," 4/28/2009, 2009; http://googleblog.blogspot.com/2008/11/now-you-can-speak-to-google-mobile-app.html.

[89] V. i. Bergl, M. \vCmejrek, M. Fanta *et al.*, "CarDialer: multi-modal in-vehicle cellphone control application," *ICMI '06: Proceedings of the 8th international conference on Multimodal interfaces*, ACM, 2006, pp. 133-134.
[90] C. L. Lisetti, and F. Nasoz, "MAUI: a multimodal affective user interface," *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, ACM, 2002, pp. 161-170.
[91] A. G. Hauptmann, R. Jin, and T. D. Ng, "Multi-modal information retrieval from broadcast video using OCR and speech recognition," *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, ACM, 2002, pp. 160-161.
[92] R. Kaucic, D. Reynard, and A. Blake, "Real-time lip trackers for use in audio-visual speech recognition." pp. 3/1-3/6.
[93] L. Chi-geun, and H. Mun-sung, "Multi-Modal Fusion of Speech-Gesture Using Integrated Probability Density Distribution." pp. 361-364.
[94] L. Ma, B. Milner, and D. Smith, "Acoustic environment classification," *ACM Trans. Speech Lang. Process.,* vol. 3, no. 2, pp. 1-22, 2006.
[95] A. Hagen, D. A. Connors, and B. L. Pellom, "The analysis and design of architecture systems for speech recognition on modern handheld-computing devices," *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ACM, 2003, pp. 65-70.
[96] C. A. Kamm, C. R. Shamieh, and S. Singhal, "Speech recognition issues for directory assistance applications," *Speech Communication,* vol. 17, no. 3-4, pp. 303 - 311, 1995.
[97] *Readings in speech recognition*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990.
[98] "What is a Multiple Listing Service (MLS)?," Oct. 22nd, 2009; http://www.realtor.org/law_and_policy/doj/mls_overview.
[99] "Alexa - The Web Information Company," Oct. 22nd, 2009; http://www.alexa.com/siteinfo/mlb.com.
[100] K.-F. Lee, H.-W. Hon, and R. Reddy, "An overview of the SPHINX speech recognition system," *Acoustics, Speech and Signal Processing, IEEE Transactions on,* vol. 38, no. 1, pp. 35-45, Jan, 1990.
[101] D. Huggins-Daines, M. Kumar, A. Chan *et al.*, "Pocketsphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices," *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, 2006, pp. I-I.
[102] L. Hui, D. Li, Y. Dong *et al.*, "A study on multilingual acoustic modeling for large vocabulary ASR." pp. 4333-4336.
[103] S. Fujimoto, "Designing low-power multiprocessor chips," *EE Times*, http://www.eetimes.com/showArticle.jhtml?articleID=197003420, 2007].
[104] B. H. Meyer, J. J. Pieper, J. M. Paul *et al.*, "Power-performance simulation and design strategies for single-chip heterogeneous multiprocessors," *Computers, IEEE Transactions on,* vol. 54, no. 6, pp. 684-697, 2005.
[105] J. C. Day, A. Janus, and J. Davis, "Computer and Internet Use in the United States: 2003," U. S. D. o. Commerce, ed., U.S. Census Bureau, 2005.
[106] N. Humberto T. Marques, L. C. D. Rocha, P. H. C. Guerra *et al.*, "Characterizing broadband user behavior," in Proceedings of the 2004 ACM workshop on Next-generation residential broadband challenges, 2004.

Appendix A: Vocabularies

This appendix details the vocabularies for each of our corpuses. First, we present a table as a reminder of the corpuses involved in our experiments:

Partition	# Words	Partition	# Words
Unified	337	Teams	30
MLB	164	Cities	27
MLS	144	MLB Nav	59
General	29	Area	88
MLB+Gen	193	Desc.	28
MLS+Gen	173	MLS Nav	34
Names	48		

Table 9: Corpus List

Note that the Unified (MLB+MLS+Gen) is the union of the MLB, MLS and General corpuses. MLB+Gen is the union of MLB and General, and MLS+Gen is the union of MLS and General. MLB is the union of Names, Teams, Cities, and MLB Navigation. MLS is the union of Area, Description, and MLS Navigation. Note that the number of words listed in the table is the count *before* our dynamic test. After giving a full breakdown of each vocabulary, we will also show the words removed and added for our dynamic test.

MLB

Active
Adam
Alberto
Anderson
Angels
April
Arizona
Astros
Athletics
Atlanta
August
Austin
Baltimore
Bard
Batter
Belliard
Bergmann
Biemel
Blue_Jays
Boston
Braves
Brewers
Calendar
Card
Cardinals
Chicago
Cincinnati
Cleveland
Colome
Colorado
Craig
Cristian
Cubs
December
Detroit
Detwiler
Diamondbacks
Dodgers
Dukes
Dunn
Eighteenth
Eighth
Eleventh
Elijah
February
Fifteenth
Fifth
Final
First
Florida
Forth
Fourteenth
Giants
Gonzalaz
Guzman
Hanrahan
Harris
Hernandez
Houston
Indians
January
Jason
Jesus
Joe
Joel
John
Johnson
Jordon
Josh
Julian
July
June
Kansas_City
Kearns
Lannan
Leaders
League
Los_Angeles
MacDougal
March
Mariners
Marlins
Martis
May

Mets
Mike
Milestone
Milwaukee
Minnesota
Nationals
New_York
Next
Nick
Nieves
Nineteenth
Ninth
November
Oakland
October
Orioles
Padres
Philadelphia
Phillies
Pirates
Pitcher
Pittsburgh
Player
Rangers
Rays
Reds
Red_Sox
Regular
Rockies
Ron
Ronnie
Ross
Roster
Royals
Ryan
San_Diego
San_Francisco
Scoreboard
Search
Season
Seattle
Second
September
Seventeenth
Seventh
Shairon
Sites
Sixteenth
Sixth
Spring
Stammen
Standings
Stats
St_Louis
Tampa_Bay
Tavarez
Team
Tenth
Texas
Third
Thirteenth
Tigers
Today
Toronto
Tracker
Training
Twelfth
Twentieth
Twins
Versus
Villone
Washington
White_Sox
Wil
Wild
Willie
Willingham
Yankees
Yesterday
Zimmerman

MLS

A
Aldie
Alexandria
Annandale
Apartment
Area
Arlington
Ashburn
B
Bathrooms
Bathroom
Bedrooms
Bedroom
Belleview
Bowling_Green
Brambleton
Bristow
Brochure
Brooke
Burke
C
Caret
Catharpin
Centreville
Champlain
Chantilly
City
Clifton
Colonial_Beach
Commercial
Condo
Co-Op
County
Crystal_City
Culpeper
D
Dahlgren
Dale_City
DC
Dulles
Dumfries
Dunn_Loring
E
Email
F
Fairfax
Fairfax_Station
Falls_Church
Falmouth
Farm
Fort_Belvoir
Fort_Myer
Franconia
Fredericksburg
G
Gainesville
Garrisonville
Great_Falls
H
Hague
Hamilton
Haymarket
Herndon
Home
Hustle
I
ID
J
K
King_George
Kingstowne
L
Lake_Ridge
Land
Lansdowne
Leesburg
Lorton
Lot
Lovettsville
M
Manassas
Manassas_Park
Map
McLean
Merrifield
Million
MLS
Mobile
Montross
Mount_Vernon
N
Nokesville

Northern_Virginia
North_Springfield
O
Oak_Hill
Oakton
Occoquan
Of
Orange
P
Paeonian_Springs
Photos
Port_Royal
Potomac
Potomac_Falls
Prince_William
Purcellville
Q
Quantico
R
Ranch
Rental
Reston
Rosslyn
Round_Hill
S
Seven_Corners
Single_Family
South_Riding
Springfield
Stafford
State
Stephens_City
Sterling
Summary
T
Townhouse
Triangle
U
V
Vienna
Virginia
W
Washington_DC
Waterford
Waterfront
West_Mclean
West_Springfield
Woodbridge
Woodford
X
Y
Z

General

Eight
Eighteen
Eighty
Eleven
Fifteen
Fifty
Five
Forty
Four
Fourteen
Hundred
Nine
Nineteen
Ninety
One
Seven
Seventeen
Seventy
Six
Sixteen
Sixty
Ten
Thirteen
Thirty
Thousand
Three
Twelve
Twenty
Two

Names

Adam
Alberto
Anderson
Austin
Bard
Belliard
Bergmann
Biemel
Colome
Craig
Cristian
Detwiler
Dukes
Dunn
Elijah
Gonzalaz
Guzman
Hanrahan
Harris
Hernandez
Jason
Jesus
Joe
Joel
John
Johnson
Jordon
Josh
Julian
Kearns
Lannan
MacDougal
Martis
Mike
Nick
Nieves
Ron
Ronnie
Ross
Ryan
Shairon
Stammen
Tavarez
Villone
Wil
Willie
Willingham
Zimmerman

Teams

Angels
Astros
Athletics
Blue_Jays
Braves
Brewers
Cardinals
Cubs
Diamondbacks
Dodgers
Giants
Indians
Mariners
Marlins
Mets
Nationals

Orioles
Padres
Phillies
Pirates
Rangers
Rays
Reds
Red_Sox
Rockies
Royals
Tigers
Twins
White_Sox
Yankees

Cities

Arizona
Atlanta
Baltimore
Boston
Chicago
Cincinnati
Cleveland
Colorado
Detroit
Florida
Houston
Kansas_City
Los_Angeles
Milwaukee
Minnesota
New_York
Oakland
Philadelphia
Pittsburgh
San_Diego
San_Francisco
Seattle
St_Louis
Tampa_Bay
Texas
Toronto
Washington

MLB Navigation

Active
April
August
Batter
Calendar
Card
December
Eighteenth
Eighth
Eleventh
February
Fifteenth
Fifth
Final
First
Forth
Fourteenth
January
July
June
Leaders
League
March
May
Milestone
Next
Nineteenth
Ninth
November
October
Pitcher
Player
Regular
Roster
Scoreboard
Search
Season
Second
September
Seventeenth
Seventh
Sites
Sixteenth
Sixth
Spring
Standings
Stats
Team
Tenth
Third
Thirteenth
Today
Tracker
Training
Twelfth
Twentieth
Versus
Wild
Yesterday

Area

Aldie
Alexandria
Annandale
Arlington
Ashburn
Belleview
Bowling_Green
Brambleton
Bristow
Brooke
Burke
Caret
Catharpin
Centreville
Champlain
Chantilly
City
Clifton
Colonial_Beach
County
Crystal_City
Culpeper
Dahlgren
Dale_City
DC
Dulles
Dumfries
Dunn_Loring
Fairfax
Fairfax_Station
Falls_Church
Falmouth
Fort_Belvoir
Fort_Myer
Franconia
Fredericksburg
Gainesville
Garrisonville
Great_Falls
Hague
Hamilton
Haymarket
Herndon
Hustle

King_George	Montross	Potomac	Stafford
Kingstowne	Mount_Vernon	Potomac_Falls	Stephens_City
Lake_Ridge	Nokesville	Prince_William	Sterling
Lansdowne	North_Springfield	Purcellville	Triangle
Leesburg	Oak_Hill	Quantico	Vienna
Lorton	Oakton	Reston	Washington_DC
Lovettsville	Occoquan	Rosslyn	Waterford
Manassas	Of	Round_Hill	West_Mclean
Manassas_Park	Orange	Seven_Corners	West_Springfield
Mclean	Paeonian_Springs	South_Riding	Woodbridge
Merrifield	Port_Royal	Springfield	Woodford

Description

Apartment	Commercial	ID	Rental
Area	Condo	Land	Single_Family
Bathrooms	Coop	Lot	State
Bathroom	County	Million	Summary
Bedrooms	Email	Mobile	Townhouse
Bedroom	Farm	Of	Virginia
City	Home	Ranch	Waterfront

MLS Navigation

A	H	N	T
Brochure	I	Northern_Virginia	U
B	ID	O	V
C	J	P	W
D	K	Photos	X
Email	L	Q	Y
E	Map	R	Z
F	MLS	S	
G	M	Summary	

Dynamic Removal

Anderson	Hanrahan	Jordon	Ross
Austin	Hernandez	Julian	Shairon
Belliard	Jesus	Kearns	Tavarez
Biemel	Joe	Martis	Zimmermann
Colome	Joel	Nick	
Detwiler	Johnson	Ronnie	

Dynamic Addition

Burnett	Clippard	Flores	Garate

Garrett	JD	Jorge	Livan
Hernandez	Jesus	Justin	Martin
Maxwell	Orr	Rivera	Sosa
Mock	Padilla	Saul	Tyler
Morse	Pete	Sean	Victor

Appendix B: Annotated List of Figures

Figure 1: Contextual Partitioning.....1

This figure depicts an architecture with resources partitioned coarsely into M contexts, C1 through Cm, each finely divided by sub-contexts. Each sub-context has a processor and memory pair assigned to it. A context activation algorithm forwards input to one or several partitions based on the situation of the user. The user's context is calculated by the activation algorithm, from contextual inputs. Contextual inputs are hard data, provided by sources such as environment sensors, websites, and user history, used to classify the user's current context

Figure 2: Task/Thread Level Parallelism 3

Illustrates TLP dividing jobs from applications into mutually exclusive tasks for parallel processing on multiple processing elements, or PEs. Most important for us is that this specialization of PEs to tasks avoids overlap in component functionality, whereas the brain is thought to embrace such overlap.

Figure 3: Coarse-Grained CP 5

Contextual partitioning applied to a speech recognition vocabulary. This figure depicts dividing a vocabulary coarsely based on broad contexts. These contexts are then assigned to independent processing elements. This leads to processing of speech input with small contextual vocabularies.

Figure 4: Fine-Grained CP 6

Contextual partitioning applied to a speech recognition vocabulary. This figure depicts dividing a vocabulary finely based on sub-contexts grouped under broader contexts. Sub-contexts are assigned independent processing elements, leading to processing of smaller vocabulary subsets than coarse-grained CP.

Figure 5: CP with Multiple Outputs.....11

Application input for our architecture is forwarded to contextual partitions by a Context Activation Algorithm (CAA). If the CAA determines multiple contexts may apply, given the user's current situation, input may be forwarded to multiple partitions resulting in multiple outputs. This situation is illustrated in this figure, where an input is forwarded to two sub-contexts, each producing an independent output.

Figure 6: HMM Topologies from [32]. 14

Depicts three common state transition topologies for HMMs. Topology of an HMM determines how many state transitions are possible from each state and is an important variable of complexity. More complex topologies, with more state connections, lead to increased training requirements. The linear topology allows each state to return to itself or progress to the state immediately following it. Left-to-Right allows each state to return to itself or to any state after it, rather than just its immediate neighbor. Bakis, allows each state to return to itself, progress to the next state, or skip the next state instead progressing to the following state. Bakis has been shown to be a successful compromise of flexibility and training, and is commonly used. Illustrated under "Fair Use" copyright guidelines from [32].

Figure 7: TLP Speedup from [38] 17

Shows speedup from TLP when applied to speech recognition from research in [38]. The authors parallelized a Hidden Markov Model speech decoding algorithm and implemented it, varying the processor count from 1 to 16. The authors noted the speedup flattens with increasing processors due to sequential aspects of the algorithm and competition for shared resources. TLP's scalability has been called into question over many algorithms, and forms the need for new levels of parallelism, in order to leverage increasing core counts on chips. Illustrated under "Fair Use" copyright guidelines.

Figure 8: Brain areas for speech perception from [45]. 19

Illustrates brain areas used for speech processing in the brain from [45]. Depicts how multiple, disparate areas are leveraged for this task, providing a demonstration of parallelism in the brain. Content illustrated under "Fair Use" copyright guidelines. Ear image licensed under Creative Commons Public Domain, available from http://openclipart.org/media/files/bionet/10903. Brain image, from http://www.freeclipartnow.com/science/medicine/anatomy/brain-2.jpg.html, licensed under Public Domain as per Terms of Use under http://www.freeclipartnow.com/termsofuse.php

Figure 9: Realtor/Baseball Fan CP 34

Depicts an example of our architecture, where resources have been assigned to a work and hobby context for an example user. The example user is a real-estate agent who follows baseball, which we define as occupation and hobby contexts, each with sub-contexts to which resources are assigned. We define a contextual input as the URL of a website currently viewed by the user, as a means to determine the user's context, and under which context the application input should be processed.

Figure 10: Realtor/Baseball Fan Coarse-Grained CP 36

Shows a coarse-grained architecture we consider in this research for our example user. Resources are split into two contexts, MLS and MLB. This architecture does not implement a general partition; therefore, overlapping vocabulary between the two contexts must be replicated into both contexts.

Figure 11: Realtor/Baseball Fan Fine-Grained CP, Hybrid Models 37

Shows a fine-grained architecture for our example user. Resources are split based on sub-contexts of our two contexts of MLS and MLB. Different partitions use different speech units based on size of vocabulary assigned. A general partition is not implemented, which could present a problem if overlapping vocabulary must be replicated across many sub-contexts.

Figure 12: Realtor/Baseball Fan Coarse-Grained CP with GP 38

Depicts a coarse-grained architecture we consider in this research for our example user when a general partition has been implemented. Now, instead of replicated overlapping vocabulary into both contexts, overlapping vocabulary is instead placed into its own separate partition.

Figure 13: Realtor/Baseball Fan Fine-Grained CP with GP 39

Depicts the fine-grained architecture we consider in this research for our example user when a general partition has been implemented. This prevents overlapping vocabulary from needing to be replicated across many sub-contexts. We believe a general partition will be necessary for most fine-grained approaches, thus this is the only fine-grained architecture we consider – ignoring fine-grained without a general partition.

Figure 14: Decoding Result Example 44

An example decoding result returned by PocketSPHINX. Depicts two types of recognition errors: insertion, where a word is hypothesized for which no word was uttered, and substitution, where an incorrect word is hypothesized for a reference word. Deletion errors, not shown, are the opposite of insertion errors, where a reference word was uttered, but no hypothesis was made.

Figure 15: Error/Samples Coarse-Grained Partitioning vs. Unified 52

Presents a plot of our results for coarse-grained partitioning against a unified architecture across the metrics of accuracy and sample count. Shows that coarse-grained uses samples more efficiently than unified, reaching lower error rates, and

that vocabulary replication in this case is more effective, in terms of accuracy and efficiency, than a general partition.

Figure 16: Error/Training Coarse-Grained Partitioning vs. Unified.....53

Presents a plot of our results for coarse-grained partitioning against a unified architecture across the metrics of accuracy and training time. Shows that coarse-grained trains more efficiently than unified, reaching lower error rates per training time consumed, and that vocabulary replication in this case is more effective, in terms of accuracy and efficiency, than a general partition.

Figure 17: Error/Samples MLB Fine-Grained vs. Coarse-Grained.....55

Presents a plot of our results for fine-grained partitioning of MLB against the coarse MLB partition across the metrics of accuracy and samples. Shows that fine-grained improves accuracy by 92% compared to coarse at similar sample counts (242 samples for fine, 244 for coarse).

Figure 18: Error/Training MLB Fine-Grained vs. Coarse-Grained.....55

Presents a plot of our results for fine-grained partitioning of MLB against the coarse MLB partition across the metrics of accuracy and training time. Shows that fine-grained improves accuracy by 88.6% compared to coarse at similar training times (70 minutes for fine, 74 minutes for coarse).

Figure 19: Error/Samples MLS Fine-Grained vs. Coarse-Grained 56

Presents a plot of our results for fine-grained partitioning of MLS against the coarse MLS partition across the metrics of accuracy and samples. No improvement found from fine-partitioning, as coarse manages a better overall accuracy.

Figure 20: Error/Training MLS Fine-Grained Vs. Coarse-Grained 57

Presents a plot of our results for fine-grained partitioning of MLS against the coarse MLS partition across the metrics of accuracy and training time. No improvement found from fine-partitioning, as coarse manages a better overall accuracy.

Figure 21: MLB/MLS Sub-Contextual Breakdown 57

Depicts a breakdown of our MLB and MLS vocabularies into sub-contextual partitions. Shows the percentage of words, percentage of total samples, and percentage of total training required by each partition.

Figure 22: Error/Samples Hybrid CP vs. Unified....63

Considers the case of leveraging fine-grained partitioning for MLB and coarse-grained for MLS, for a hybrid partitioned approach. Presents a plot of our results for hybrid partitioning against unified across the metrics of accuracy and samples. The best accuracy of Hybrid is 61% higher than the best case of unified, taking advantage of the benefits of fine-grained partitioning of MLB, and coarse-grained of MLS.

Figure 23: Error/Training Hybrid CP vs. Unified....64

Considers the case of leveraging fine-grained partitioning for MLB and coarse-grained for MLS, for a hybrid partitioned approach. Presents a plot of our results for hybrid partitioning against unified across the metrics of accuracy and training. The best accuracy of Hybrid is 61% higher than the best case of unified, taking advantage of the benefits of fine-grained partitioning of MLB, and coarse-grained of MLS.

Figure 24: TLP Speedup for SR from [38]....66

Reprint of Figure 7.

Figure 25: Ideal CLP Speedup vs. TLP Speedup from [38]....67

Adds CLP's ideal speedup to Figure 7. Shows that a vocabulary divisible evenly by M contexts assigned to M cores provides linear speedup proportional to M. Although fixed vocabularies can only be subdivided so many times, limiting M, a real-time vocabulary can use addition context/core assignments to add vocabulary via new contexts, leading to linear increases in vocabulary size – ignoring certain overheads.

Figure 26: Word Acoustic Model Sizes....68

Compares acoustic model sizes for our word unit trained partitions. Shows proportional increase in model size relative to number of words in the partition. If vocabulary for word trained partitions is limited, small acoustic models are possible, potentially allowing for caching of an active acoustic model for the decoding process, providing greater speedup.

Figure 27: Tri-phone Acoustic Model Sizes....69

Compares acoustic model sizes for our tri-phone unit trained partitions. Shows size becomes less dependent on word count, and is instead constrained by the number of senones used. Larger vocabularies will tend to use greater senones, leading to larger acoustic models. This supports our usage of fine-grained partitioning so as to leverage small, word-trained models for potential caching.