Designing Future Warehouse-Scale Computers for Sirius, an End-to-End Voice and Vision Personal Assistant

JOHANN HAUSWALD, MICHAEL A. LAURENZANO, YUNQI ZHANG, HAILONG YANG, YIPING KANG, CHENG LI, AUSTIN ROVINSKI, ARJUN KHURANA, RONALD G. DRESLINSKI, TREVOR MUDGE, VINICIUS PETRUCCI, LINGJIA TANG, and JASON MARS, Clarity Lab, University of Michigan at Ann Arbor; Beihang University

As user demand scales for intelligent personal assistants (IPAs) such as Apple's Siri, Google's Google Now, and Microsoft's Cortana, we are approaching the computational limits of current datacenter (DC) architectures. It is an open question how future server architectures should evolve to enable this emerging class of applications, and the lack of an open-source IPA workload is an obstacle in addressing this question. In this article, we present the design of Sirius, an open end-to-end IPA Web-service application that accepts queries in the form of voice and images, and responds with natural language. We then use this workload to investigate the implications of four points in the design space of future accelerator-based server architectures spanning traditional CPUs, GPUs, manycore throughput co-processors, and FPGAs. To investigate future server designs for Sirius, we decompose Sirius into a suite of eight benchmarks (Sirius Suite) comprising the computationally intensive bottlenecks of Sirius. We port Sirius Suite to a spectrum of accelerator platforms and use the performance and power trade-offs across these platforms to perform a total cost of ownership (TCO) analysis of various server design points. In our study, we find that accelerators are critical for the future scalability of IPA services. Our results show that GPU- and FPGA-accelerated servers improve the query latency on average by 8.5× and 15×, respectively. For a given throughput, GPU- and FPGA-accelerated servers can reduce the TCO of DCs by 2.3× and 1.3×, respectively.

CCS Concepts: • **Computer systems organization → Architectures;**

Additional Key Words and Phrases: Datacenters, warehouse-scale computers, emerging workloads, intelligent personal assistants

ACM Reference Format:
Johann Hauswald, Michael A. Laurenzano, Yunqi Zhang, Hailong Yang, Yiping Kang, Cheng Li, Austin Rovinski, Arjun Khurana, Ronald G. Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, and Jason Mars. 2016. Designing future warehouse-scale computers for Sirius, an end-to-end voice and vision personal assistant. ACM Trans. Comput. Syst. 34, 1, Article 2 (April 2016), 32 pages.
DOI: http://dx.doi.org/10.1145/2870631

This article extends the version published at ASPLOS 2015. This work was partially sponsored by Google, ARM, the Defense Advanced Research Projects Agency (DARPA) under agreement HR0011-13-2-000, and the National Science Foundation (NSF) under grants CCF-SHF-1302682 and CNS-CSR-1321047.
The work of H. Yang was conducted as a postdoctoral fellow of Clarity Lab at the University of Michigan.
Authors' addresses: J. Hauswald, M. A. Laurenzano, Y. Zhang, Y. Kang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. Mudge, V. Petrucci, L. Tang, and J. Mars (corresponding author), Computer Science and Engineering Department, University of Michigan at Ann Arbor; emails: {jahausw, mlaurenz, yunqi, ypkang, elfchris, rovinski, khuranaa, rdreslin, tnm, lingjia, profmars}@umich.edu; H. Yang, Baihang University; email: hailong@umich.edu.

DOI: http://dx.doi.org/10.1145/2870631

1. INTRODUCTION

Apple's Siri [AppleSiri2011], Google's Google Now [GoogleNow 2014], and Microsoft's Cortana [MicrosoftCortana 2015] represent a class of emerging Web-service applications known as intelligent personal assistants (IPAs). An IPA is an application that uses inputs such as the user's voice, vision (images), and contextual information to provide assistance by answering questions in natural language, making recommendations, and performing actions. These IPAs are emerging as one of the fastest-growing Internet services, as they have recently been deployed on well-known platforms such as iOS, Android, and Windows Phone, making them ubiquitous on mobile devices worldwide [IDCMobile 2015]. In addition, the usage scenarios for IPAs are rapidly increasing with recent offerings in wearable technologies such as smart watches [GoogleAndroidWear 2014] and smart glasses [GoogleGlass 2014]. Recent projections predict the wearables market to be at 485 million annual device shipments by 2018 [ABIResearch 2013]. This growth in market share, coupled with the fact that the design of wearables is heavily reliant on voice and image input, further indicates that rapid growth in user demand for IPA services is on the horizon.

IPAs differ from many of the Web-service workloads currently present in modern warehouse-scale computers (WSCs). In contrast to the queries of traditional browser-centric services, IPA queries stream through software components that leverage recent advances in speech recognition, natural language processing (NLP), and computer vision to provide a speech-driven and/or image-driven contextually based question-and-answer system to users [Hearst 2011]. Due to the computational intensity of these components and the large data-driven models they use, service providers house the required computation in massive datacenter (DC) platforms in lieu of performing the computation on the mobile devices themselves. This offloading approach is used by both Apple's Siri and Google's Google Now, as they send compressed recordings of voice command/queries to DCs for speech recognition and semantic extraction [Siegler 2011]. However, DCs have been designed and tuned for traditional Web services such as Web Search (WS), and questions arise as to whether the current design employed by modern DCs, composed of general-purpose servers, is suitable for emerging IPA workloads.

IPA queries require a significant amount of compute resources compared to traditional text-based Web services such as WS. As we show later in this work, the computational resources required for a single leaf query is in excess of 100× more than that of traditional WS. Figure 1 illustrates the scaling of compute resources in a modern DC required to sustain an equivalent throughput of IPA queries compared to WS. Due to the looming *scalability gap* shown in the figure, there has been significant interest in both academia and industry to leverage hardware acceleration in DCs using various platforms such as GPU, manycore co-processors, and FPGAs to achieve high performance and energy efficiency. To gain further insight on whether there are sufficient acceleration opportunities for IPA workloads, as well as identification of the best acceleration platform, several challenges need to be addressed, including the following:

(1) Identifying critical compute and performance bottlenecks throughout the end-to-end lifetime of an IPA query
(2) Understanding the performance, energy, and cost trade-offs among popular accelerator options given the characteristics of IPA workloads
(3) Designing future server and DC solutions that can meet the amount of future user demand while being cost and energy efficient.

However, the lack of a representative, publicly available, end-to-end IPA system proves prohibitive for investigating the design space of future accelerator-based server designs for this emerging workload. To address this challenge, we first construct an end-to-end

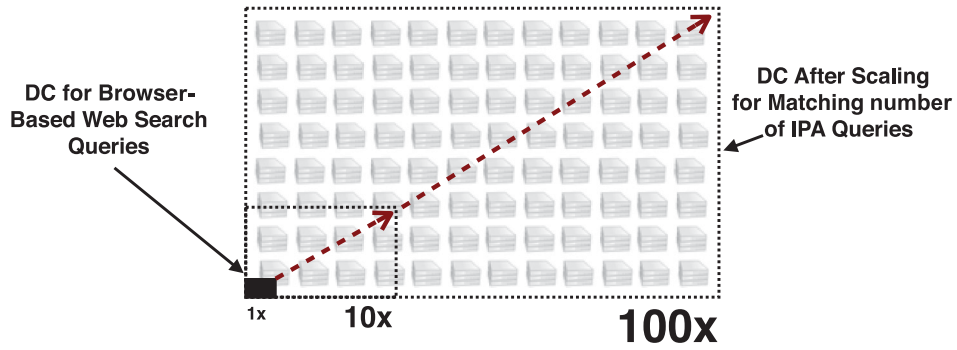


Fig. 1. Impact of higher computational requirements for IPA queries on DCs.

stand-alone IPA service—Sirius—that implements the core functionalities of an IPA such as speech recognition, image matching, NLP, and a question-and-answer system. Sirius takes as input user-dictated speech and/or image(s) captured by a camera. There are three pathways of varying complexity through the Sirius back-end based on the nature of the input query. A voice command primarily exercises speech recognition on the server side to execute a command on the mobile device. A voice query additionally leverages a sophisticated NLP question-and-answer system to produce a natural language response to the user. A voice and image question such as *When does this restaurant close?* coupled with an image of the restaurant also leverages image matching with an image database and combines the matching output with the voice query to select the best answer for the user. We have constructed Sirius by integrating three services built using well-established open-source projects that include techniques and algorithms representative of those found in commercial systems. These open projects include CMU's Sphinx [Huggins-Daines et al. 2006], representing the widely used Gaussian mixture model (GMM)-based speech recognition; Kaldi [Povey et al. 2011], and RWTH's RASR [Rybach et al. 2011], representing industry's recent trend toward deep neural network (DNN)-based speech recognition; OpenEphyra (OE) [Seide et al. 2011], representing the state-of-the-art question-and-answer system based on IBM's Watson [Ferrucci et al. 2010]; and SURF [Bay et al. 2006], implemented using OpenCV [Bradski 2000] and representing state-of-the-art image matching algorithms widely used in various production applications.

With this end-to-end workload in hand, we perform an in-depth investigation of the viability of various acceleration strategies and provide insights on future DC and server designs for this emerging workload. Specifically, our work makes the following contributions:

—*Sirius*: We construct Sirius, an open end-to-end IPA system with both speech and image front-ends. In addition to Sirius itself, we compile a query taxonomy spanning three classes of queries: Voice Command (VC), Voice Query (VQ), and Voice-Image Query (VIQ) (Section 2).
—*Scalability gap*: We characterize Sirius on commodity hardware and demonstrate the scalability gap for this type of workload. We observe that the compute resources needed to sustain this workload is orders of magnitude higher than traditional DC workloads. We also perform an analysis of the cycle breakdown of IPA queries and analyze the computational bottlenecks of Sirius. We show that there is a limited speedup potential for this workload on general-purpose processors and that acceleration is indeed needed to address the scalability gap (Section 3).

—*Accelerating Sirius*: Based on our cycle breakdown analysis, we extract seven computational bottlenecks comprising 92% of the cycles consumed by Sirius to compose a C/C++ benchmark suite (Sirius Suite) for acceleration. We port these workloads and conduct a thorough performance evaluation on a spectrum of accelerator platforms. The end-to-end Sirius, query taxonomy, input set, Sirius Suite benchmarks, and the full source code ported across accelerators are available online [ClarityLab 2015] (Section 4).
—*Future server and DC design*: Based on our acceleration results, we investigate the implications for future server designs. After evaluating the trade-offs between performance, power efficiency, and the total cost of ownership (TCO) of a DC, we propose server and DC designs that significantly reduce the computation gap between user demand and the current DC's computation capability (Section 5).
—*Extending Sirius*: We extend both the Sirius application and benchmark suite to incorporate the Object Recognition (OR) service, which represents the intelligent user queries relying on the cutting-edge computer vision techniques. We also conduct a thorough analysis on the bottlenecks of object recognition and evaluate the latency, energy efficiency, and TCO when ported to different accelerator platforms (Section 6).

In summary, we find that among the popular acceleration options, including GPU, Intel Phi, and FPGA, the FPGA-accelerated server is the best server option for a homogeneous DC design when the design objective is minimizing latency or maximizing energy efficiency with a latency constraint. FPGA achieves an average 18× reduction on the query latency across various query types over the baseline multicore system. On the other hand, GPUs provide the highest TCO reduction on average. GPU-accelerated servers can achieve an average 8.5× query latency reduction, translating to a 2.3× TCO reduction. When excluding FPGAs as an acceleration option, GPUs provide the best latency and cost reduction among the rest of the accelerator choices. On average, replacing FPGAs using GPUs leads to a 76% longer latency but in return achieves a 43% TCO reduction and simpler software engineering costs.

2. SIRIUS: AN END-TO-END IPA

In this section, we present Sirius: an end-to-end IPA (IPA). We first describe the design objectives for Sirius and then present an overview of Sirius and a taxonomy of query types that it supports. Finally, we detail the underlying algorithms and techniques used by Sirius.

2.1. Sirius Design Objectives

There are three key objectives in the design for Sirius:

(1) *Completeness*: Sirius should provide a complete IPA service that takes the input of human voice and images and provide a response to the user's question with natural language.
(2) *Representativeness*: The computational techniques used by Sirius to provide this response should be representative of state-of-the-art approaches used in commercial domains.
(3) *Deployability*: Sirius should be deployable and fully functional on real systems.

2.2. Sirius Overview: Life of an IPA Query

Figure 2 presents a high-level diagram of the end-to-end Sirius query pipeline. The life of a query begins with a user's voice and/or image input through a mobile device. Compressed versions of the voice recording and image(s) are sent to a server housing Sirius.

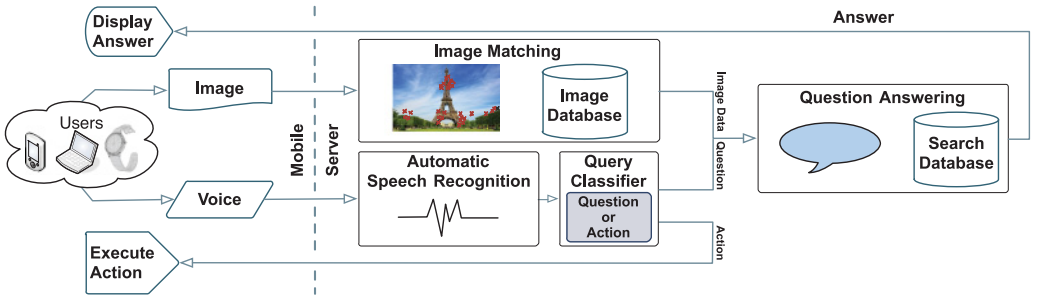


Fig. 2. End-to-end diagram of the Sirius pipeline.

Table I. Query Taxonomy

Query Type	Properties	
Voice Command (VC)	**Example**	"Set my alarm for 8am."
	Service	ASR
	Result	Action on user's device
	Queries (#)	16
Voice Query (VQ)	**Example**	"Who was elected 44th president?"
	Service	ASR & QA
	Result	Best answer from QA
	Queries (#)	16
Voice-Image Query (VIQ)	**Example**	"When does this restaurant close?"
	Service	ASR, IMM, & QA
	Result	Best results from IMM & QA
	Queries (#)	10

The user's voice is then processed by the Automatic Speech Recognition (ASR) front-end, which translates the user's speech question into its text equivalent using statistical models. The translated speech then goes through a query classifier (QC) that decides if the speech is an action or a question. If it is an action, the command is sent back to the mobile device for execution. Otherwise, the Sirius back-end receives the question in plain text. Using NLP techniques, the Question Answering (QA) service extracts information from the input, searches its database, and chooses the best answer to return to the user. If an image accompanies the speech input, Sirius uses computer vision techniques to attempt to match the input image to its image database and return relevant information about the matched image using the Image Matching (IMM) service. For example, a user can ask *What time does this restaurant close?* while image(s) of the restaurant are captured via smart glasses [GoogleGlass 2014]. Sirius can then return an answer to the query based not only on the speech but also on information from the image.

As shown in Figure 2, there are several pathways a single query can take based on the type of directive, whether it be question or action, and the type of input, either speech only or accompanied by images. To design the input set used with Sirius, we have identified a query taxonomy of three classes that covers these pathways. Table I summarizes these query classes providing an example for each, the Sirius services they exercise, the resulting behavior of Sirius, and the number of queries of that type in our input set.

Figure 3 illustrates a tiered view of Sirius spanning the query taxonomy it supports, the services that comprise Sirius, and the algorithmic subcomponents that compose each service. We describe these services and algorithms in the following section.

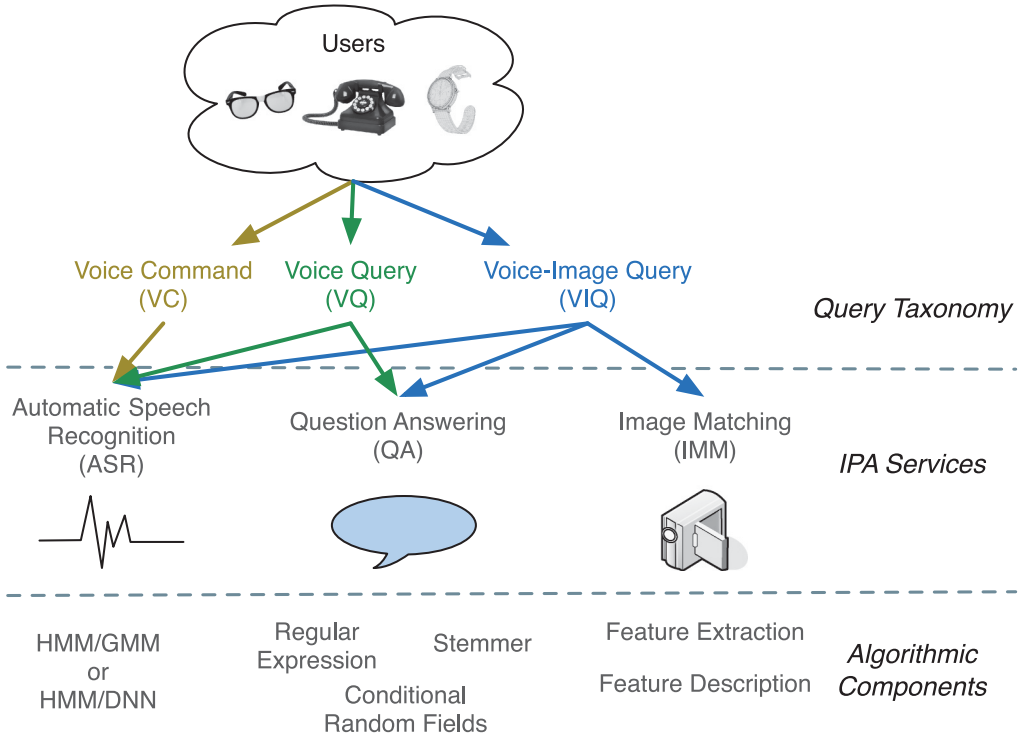


Fig. 3. Tier-level view of Sirius.

2.3. The Design of Sirius: IPA Services and Algorithmic Components

As shown in Figure 3, Sirius is composed of three IPA services: ASR, QA, and IMM. These services can be further decoupled into their individual algorithmic components. To design Sirius to be representative of production grade systems, we leverage well-known open infrastructures that use the same algorithms as commercial applications. Speech recognition in Google Voice, for example, has used the speaker-independent GMM and hidden Markov model (HMM) and is adopting DNNs [Hinton et al. 2012; Dean et al. 2012]. The OE framework used for QA is an open-source release from CMU's prior research collaboration with IBM on the Watson system [Ferrucci et al. 2010]. OE's NLP techniques, including conditional random fields (CRF), have been recognized as the state of the art and are used at Google and in other industry QA systems [Tackstrom et al. 2013]. We design our image matching pipeline based on the SURF algorithm, which is widely used in industry [Bay et al. 2006; MobileMarketing 2014]. We implement SURF using the Open Source Computer Vision Library (OpenCV) [Bradski 2000], which is employed in commercial products from companies like Google, IBM, and Microsoft. The design of these services are described in the remainder of this section.

2.3.1. Automatic Speech Recognition. The inputs to the ASR are feature vectors representing the speech segment, generated by fast preprocessing and feature extraction of the speech. The ASR component relies on a combination of a HMM and either a GMM or a DNN. Sirius' GMM-based ASR uses CMU's Sphinx [Huggins-Daines et al. 2006], whereas the DNN-based ASR includes Kaldi [Povey et al. 2011] and RWTH's RASR [Rybach et al. 2011].

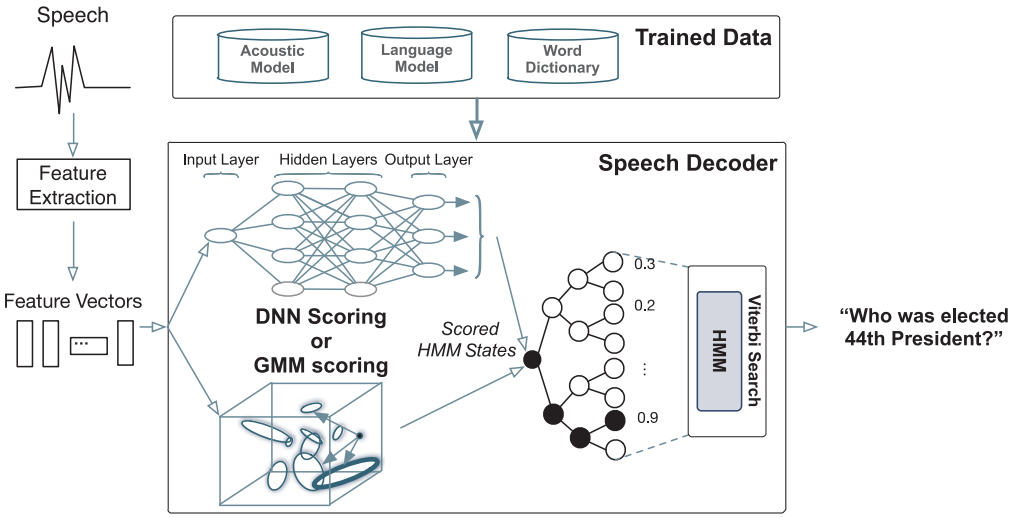


Fig. 4. ASR pipeline.

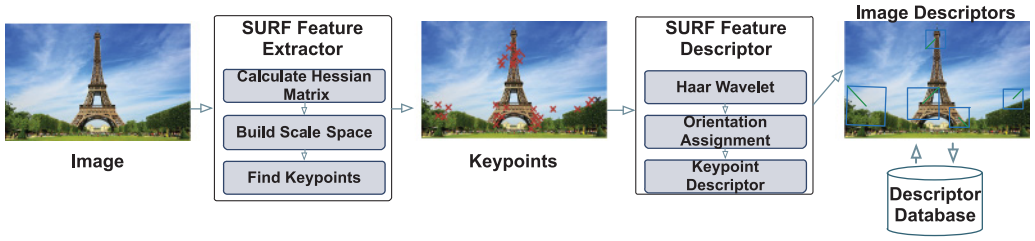


Fig. 5. Image matching pipeline.

As shown in Figure 4, the HMM builds a tree of states for the current speech frame using input feature vectors. The GMM or DNN scores the probability of the state transitions in the tree, and the Viterbi algorithm [Forney 1973] then searches for the most likely path based on these scores. The path with the highest probability represents the final translated text output. The GMM scores HMM state transitions by mapping an input feature vector into a multidimensional coordinate system and iteratively scores the features against the trained acoustic model.

DNN, however, scores using probabilities from a neural network. The depth of a DNN is defined by the number of hidden layers where scoring amounts to one forward pass through the network. In recent years, industry and academia have moved toward DNN over GMM due to its higher accuracy [Dahl et al. 2012; Huang et al. 2014].

2.3.2. Image Matching. The image matching pipeline receives an input image, attempts to match it against images in a preprocessed image database, and returns information about the matched images. The database that is used in Sirius is the Mobile Visual Search [Chandrasekhar et al. 2011] database. Image keypoints are first extracted from the input image using the SURF algorithm [Bay et al. 2006]. In Feature Extraction (FE), the image is downsampled and convolved multiple times to find interesting points at different scales. After thresholding the convolution responses, the local maxima responses are stored as image keypoints. Figure 5 details the steps in this process. The keypoints are then passed to the Feature Descriptor (FD) component, where they are assigned an

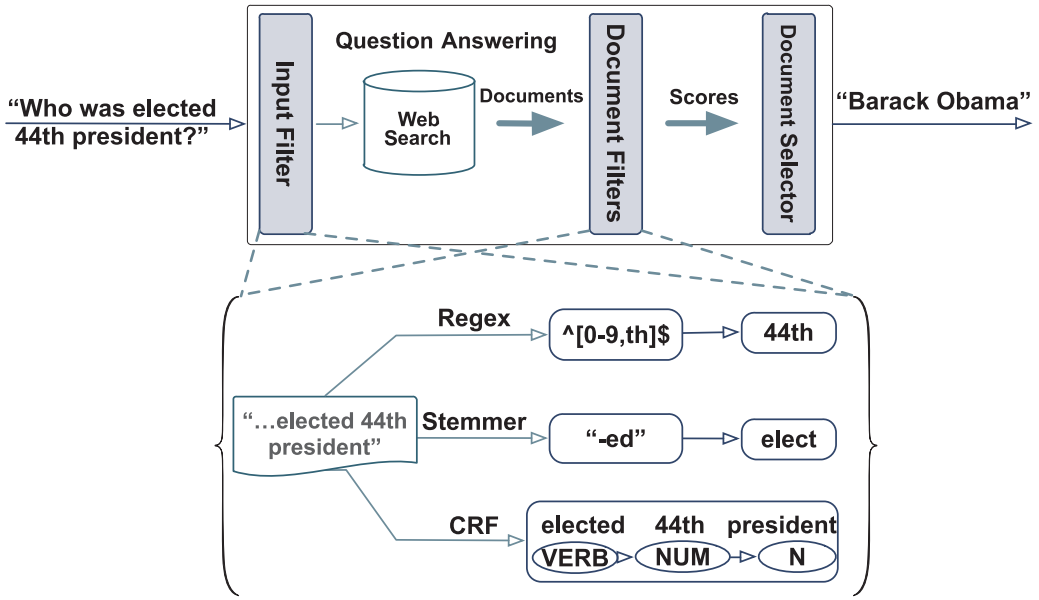


Fig. 6. OE QA pipeline.

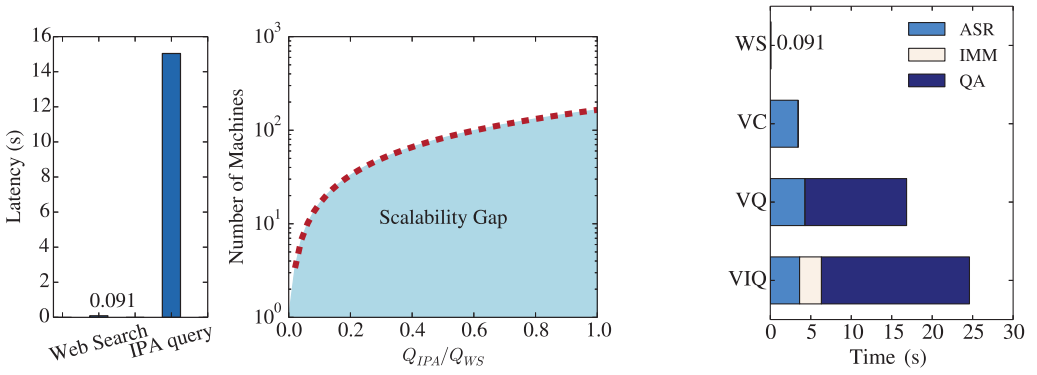
orientation vector, and similarly oriented keypoints are grouped into feature descriptors. This process reduces variability across input images, increasing chances of finding the correct match. The descriptors from the input image are matched to preclustered descriptors representing the database images by using an approximate nearest neighbor (ANN) search; the database image with the highest number of matches is returned.

2.3.3. Question Answering. The text output from the ASR is passed to OpenEphyra (OE) [Seide et al. 2011], which uses three core processes to extract textual information: word stemming, regular expression matching, and part-of-speech tagging. Figure 6 shows a diagram of the OE engine incorporating these components, generating WS queries and filtering the returned results. The Porter stemming [Porter 1980] algorithm (stemmer) exposes the root of a word by matching and truncating common word endings. OE also uses a suite of regular expression patterns to match common query words (what, where, etc.) and filter any special characters in the input. The CRF classifier [Lafferty et al. 2001] takes a sentence, the position of each word in the sentence, and the label of the current and previous word as input to makes predictions on the part of speech for each word of an input query. Each input query is parsed using the aforementioned components to generate queries to the search engine. Next, filters using the same techniques are used to extract information from the returned documents; the document with the highest overall score after score aggregation is returned as the best answer.

3. REAL SYSTEM ANALYSIS FOR SIRIUS

In this section, we present a real system analysis of Sirius. The experiments throughout this section are performed using an Intel Haswell server (details are shown later in Table III).

Scalability gap. To gain insights on the required resource scaling for IPA queries in modern DCs, we juxtapose the computational demand of an average Sirius query with that of an average WS query. To perform this experiment, we compare the average



(a) Scalability Gap

(b) Latency Across Query Types

Fig. 7. Scalability gap and latency.

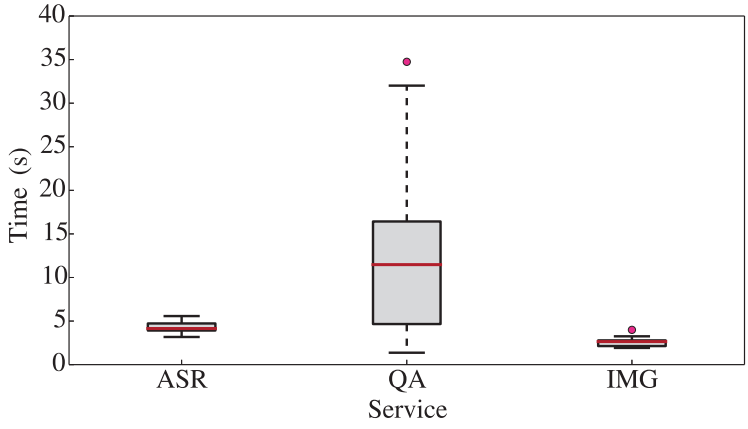
query latency (execution time) for both applications on a single core at a very low load. Both Sirius and WS are configured to be memory resident and go no further than main memory to process a query (i.e., minimum I/O activities).

Figure 7(a) (left) presents the average latency of both WS using open-source Apache Nutch [ApacheNutch 2010], Ferdman et al. [2012], and Sirius queries. As shown in the figure, the average Nutch-based WS query latency is 91ms on the Haswell-based server. In contrast, Sirius query latency is significantly longer, averaging around 19s across 42 queries spanning our three query classes (VC, VQ, and VIQ from Table I). Based on this significant difference in the computational demand, we perform a back-of-the-envelope calculation of how the compute resources (machines) in current DCs must scale to match the throughput in queries for IPAs and WS.

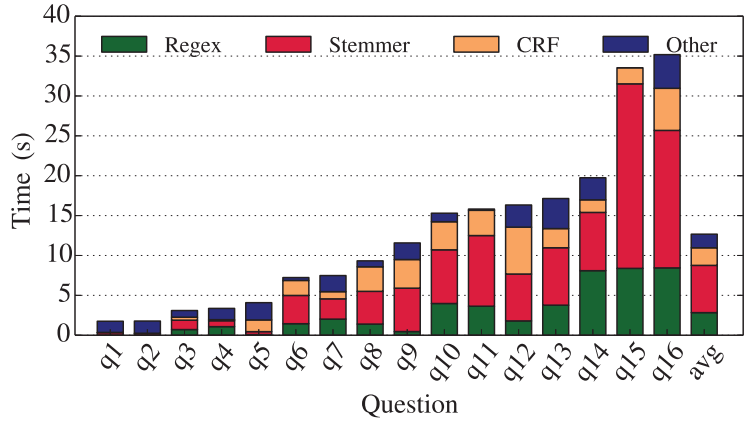
Figure 7(a) (right) presents the number of machines needed to support IPA queries as the number of these queries increases. The x-axis shows the ratio between IPA queries and traditional WS queries. The y-axis shows the ratio of compute resources needed to support IPA queries relative to WS queries. As shown in the figure, current DC infrastructures will need to scale its compute resources to 210× its current size when the number of IPA queries scale to match the number of WS queries. We refer to this throughput difference as the scalability gap.

Sirius query deep dive. To better understand the IPA query characteristics, we further investigate the average latency and latency distributions of various query types for Sirius. Figure 7(b) presents the average latency across query types including traditional WS, VC, VQ, and VIQ. As shown in the figure, the latency of all three Sirius query types are significantly higher than that of WS queries. The shortest query type is VC, which only uses the ASR service. Yet it still requires orders of magnitude more computation than WS. The longest query type is VIQ, which uses all three services including ASR, IMM, and QA. Among all three services, QA consistently consumes the most compute cycles.

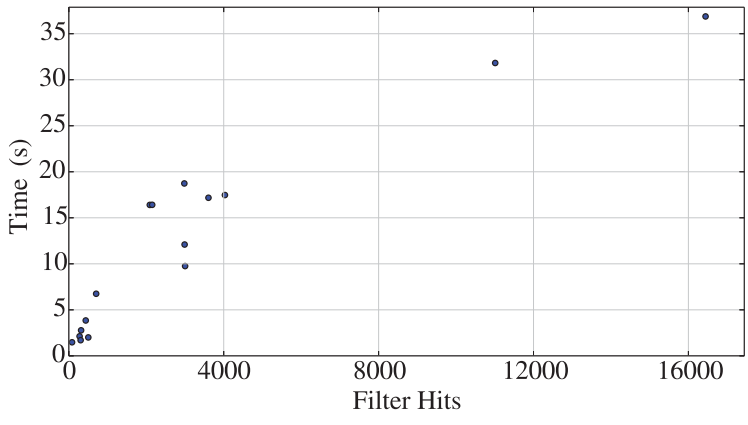
Figure 8(a) presents the latency distribution for each Sirius service. As shown in the figure, QA has the highest variability in latency, ranging from 1.7s to 35s depending on the input query. Figure 8(b) further presents the breakdown of execution time among QA's hot components (described later in this section) across the complete VQ query input set (shown in Table II). The reason for this high latency variability is not immediately clear from inspecting the query input set, especially when considering the



(a) Latency Variability Across Services



(b) OpenEphyra Breakdown



(c) Latency and Filter Hits in OpenEphyra

Fig. 8. Sirius variability across query types and causes.

Table II. Voice Query Input Set

Query Number	Query
q1	"Where is Las Vegas?"
q2	"What is the capital of Italy?"
q3	"Who is the author of Harry Potter?"
...	...
q15	"What is the capital of Cuba?"
q16	"Who is the current president of the United States?"

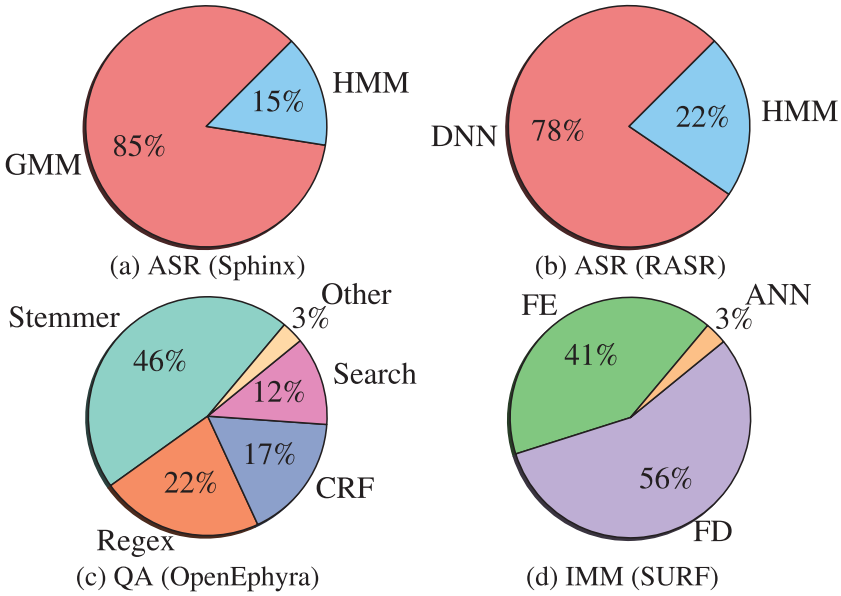


Fig. 9. Cycle breakdown per service.

small difference between Q2 and Q15 in Table II. However, after further investigation, we identified that the high variance is primarily due to the runtime variability of various document filters in the NLP component used to select the most fitting answer for a given query. Figure 8(c) demonstrates the correlation between latency and the number of hits in the document filters. The other services, ASR and IMM, have very low query to query variability. Next, we investigate the cycle breakdown of the algorithmic components that comprise each service.

Cycle breakdown of Sirius services. To identify the computational bottlenecks of each service, we perform top-down profiling of hot algorithmic components for each service, shown in Figure 3, using Intel VTune [IntelVTune 2015]. Figure 9 presents the average cycle breakdown results. Across services, a few hot components emerge as good candidates for acceleration. For example, a high percentage of the execution for ASR is spent on scoring using either GMM or DNN. For QA, on average, 85% of the cycles are spent in three components including stemming, regular expression pattern matching, and CRF; for IMM, the majority of cycles are spent either performing feature extraction or description using the SURF algorithm.

We then identify the architectural bottlenecks for these hot components to investigate the performance improvement potential for a general-purpose processor. Figure 10 presents the instructions per cycle (IPC) and potential architectural bottlenecks

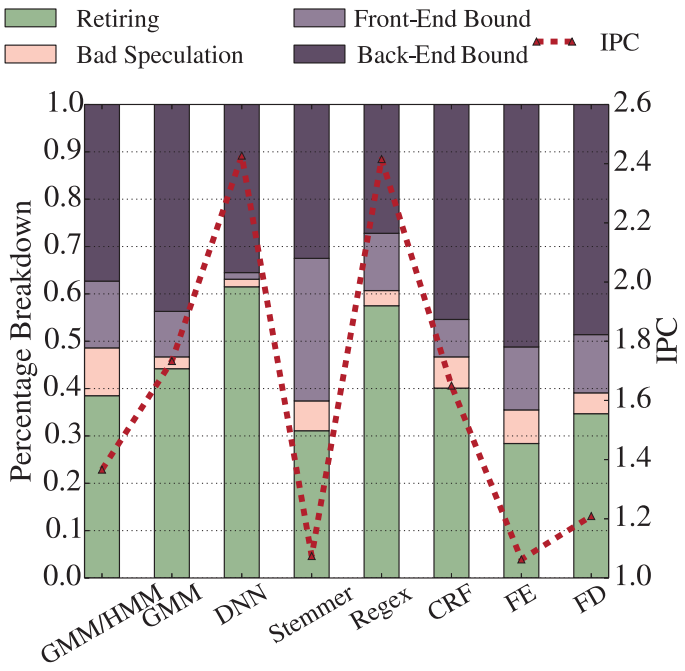


Fig. 10. IPC and bottleneck breakdown.

(including front-end, speculation, and back-end) for each component, identified using Intel VTune [IntelVTune 2015]. A few of the service components, including DNN and Regex, execute relatively efficiently on Xeon cores. This graph indicates that even with all stall cycles removed (perfect branch prediction, infinite cache, etc.), the maximum speedup is bound by around 3×. Considering the orders of magnitude difference indicated by the scalability gap, further acceleration is needed to bridge the gap.

4. ACCELERATING SIRIUS

In this section, we describe the platforms and methodology used to accelerate the key components of Sirius. We also present and discuss the results of accelerating each of these components across four different accelerator platforms.

4.1. Accelerator Platforms

We use a total of four platforms, summarized in Table III, to accelerate Sirius. Our baseline platform is an Intel Xeon Haswell CPU running single-threaded kernels.

We summarize the advantages and disadvantages of each accelerator platform next:

—Multicore CPU. *Advantages*: High clock frequency, not limited by branch divergence. *Disadvantages*: Least amount of threads available.
—GPU. *Advantages*: Massively parallel. *Disadvantages*: Power hungry, custom ISA, hard to program, large data transfer overheads, limited branch divergence handling.
—Intel Phi. *Advantages*: Many core, standard programming model (same ISA), manual porting optional/compiler help, handles branch divergence, high bandwidth. *Disadvantages*: Data transfer overheads, relies on compiler. *Note*: One core is used for the operating system running on the device itself.

Table III. Platform Specifications

Model	Frequency	Cores (#)	HW Threads (#)	Memory	Memory Bandwidth	Peak TFLOPS
Intel Xeon E3-1240 V3	3.40GHz	4	8	12GB	25.6GB/s	0.5
NVIDIA GTX 770	1.05GHz	8*	12,288	2GB	224GB/s	3.2
Intel Xeon Phi 5110P	1.05GHz	60	240	8GB	320GB/s	2.1
Xilinx Virtex-6 ML605	400MHz	N/A	N/A	512MB	6.40GB/s	0.5

*Core = SM (streaming multiprocessor), 2,048 threads/SM.

Table IV. Sirius Suite and Granularity of Parallelism

Benchmark	Baseline	Input Set	Data Granularity
Gaussian Mixture Model (GMM)[a]	CMU Sphinx	HMM states	HMM state
Deep Neural Network (DNN)[a]	RWTH RASR	HMM states	Matrix multiplication
Porter Stemming (Stemmer)[b]	Porter	4M word list	Individual word
Regular Expression (Regex)[b]	SLRE	100/400 expr./sent.	Regex-sentence pair
Conditional Random Fields (CRF)[b]	CRFsuite	CoNLL-2000	Sentence
Feature Extraction (FE)[c]	SURF	JPEG image	Image tile
Feature Description (FD)[c]	SURF	Vector of keypoints	Keypoint

[a]Extracted from ASR service: CMU Sphinx [Huggins-Daines et al. 2006], RWTH RASR [Rybach et al. 2011].
[b]Extracted from QA service: Porter [1980], SLRE [Lyubka 2009], CRFsuite [Okazaki 2007], CoNLL-2000 [Tjong et al. 2000].
[c]Extracted from IMM service: SURF [Bay et al. 2006].

—FPGA. *Advantages*: Can be tailored to implement very efficient computation and data layout for the workload. *Disadvantages*: Runs at a much lower clock frequency, expensive, hard to develop for and maintain with software updates.

4.2. Sirius Suite: A Collection of IPA Compute Bottlenecks

To investigate the viability and trade-offs of accelerating IPAs, we extract the key computational bottlenecks of Sirius (described in Section 3) to construct a suite of benchmarks that we call *Sirius Suite*. Sirius Suite and its implementations across the described accelerator platforms are available alongside the end-to-end Sirius application [ClarityLab 2015]. As a basis for Sirius Suite, we port existing open-source C/C++ implementations available for each algorithmic component to our target platforms. We additionally implemented stand-alone C/C++ benchmarks based on the source code of Sirius where none were currently available. The baseline implementations are summarized in column 2 of Table IV. For each Sirius Suite benchmark, we built an input set representative of IPA queries. The table shows the granularity at which each thread performs the computation on the accelerators. For example, both GMM and DNN kernels receive input feature vectors from the HMM search, which are all scored in parallel but at different levels of abstraction, respectively, based on each implementation.

4.3. Porting Methodology

The common porting methodology used across all platforms is to exploit the large amount of data-level parallelism available throughout the processing of a single IPA query. We describe the platform-specific highlights of our porting efforts in the following sections.

4.3.1. Multicore CPU. We use the Pthread library to accelerate the kernels on the multicore platform by dividing the size of the data. Each thread is responsible for a range of data over a fixed number of iterations. This approach allows each thread to run concurrently and independently, synchronizing only at the end of the execution.

For the image matching kernels, we preprocess the input images for feature extraction by tiling the images. Each thread of the CPU is assigned one or more tiles of the input image (depending on the size of each tile). This allows us to spawn threads once at the beginning of execution and synchronize threads at the end instead of parallelizing at a smaller granularity within the SURF algorithm, which would require multiple synchronizations between loops. However, as the tile size decreases, the number of "good" keypoints decreases, so we fix the tile size to a minimum of 50×50 per thread.

4.3.2. GPU. We use NVIDIA's CUDA library to port the Sirius components to the NVIDIA GPU. To implement each CUDA kernel, we varied and configured the GPU block and grid sizes to achieve high resource utilization, matching the input data to the best thread layout. We ported additional string manipulation functions currently not supported in CUDA for the stemmer kernel.

4.3.3. Intel Phi. We port our Pthread versions to the Intel Phi platform, leveraging the ability of the target compiler to parallelize the loops on the target platform. For this, we use Intel's ICC cross-compiler. The Phi kernel is built and run directly on the target device, allowing for rapid prototyping and debugging. On the Phi platform, we sweep the total amount of threads spawned in increments of 60, increasing the number of hardware threads per core. For some kernels, the maximum number of threads (with enough input data) did not always yield the highest performance. To investigate the potential of this platform to facilitate ease of programming, we use the standard programming model and custom compiler to extract performance from the platform. As such, the results represent what can be accomplished with minimal programmer effort.

4.3.4. FPGA. We use previously published details of FPGA implementations for several of our Sirius benchmarks in this work. However, due to limited published details for two of our workloads and to gain further insights, we design our own FPGA implementations for both GMM and Stemmer and evaluate them on a Xilinx FPGA.

GMM. The major computation of the algorithm lies in three nested loops that iteratively score the feature vector against the training data. This training data comes from an acoustic model, a language model, and a dictionary in the forms of a means vector, a precalculated (precs) vector, a weight vector, and a factor vector. All of this data is used to generate a score for the probability of an HMM state transition. Our focus when implementing the algorithm on the FPGA was to maximize parallelization and pipeline utilization, which led to the design presented in Figure 11. This figure depicts both a core that computes the score of a single iteration of the outermost loop and a callout of a log differential unit. The log differential unit is used to fully parallelize the innermost loop, whereas the entire core can be instantiated multiple times to parallelize the outermost loop. Because of this, the design is highly scalable, as multiple cores can be used to fill the FPGA fabric. The middle loop of the algorithm was not parallelizable, however, and is represented by the Log Summation unit. With this design, we were able to create a high throughput device with a linear pipeline.

Stemmer. The Stemmer algorithm computes the root of a word by checking for multiple conditions, such as the word's suffixes or roots. Figure 12 summarizes a single step for our stemmer implementation. By taking advantage of the mutual exclusivity of test conditions, we were able to parallelize these comparisons, which allowed the FPGA to

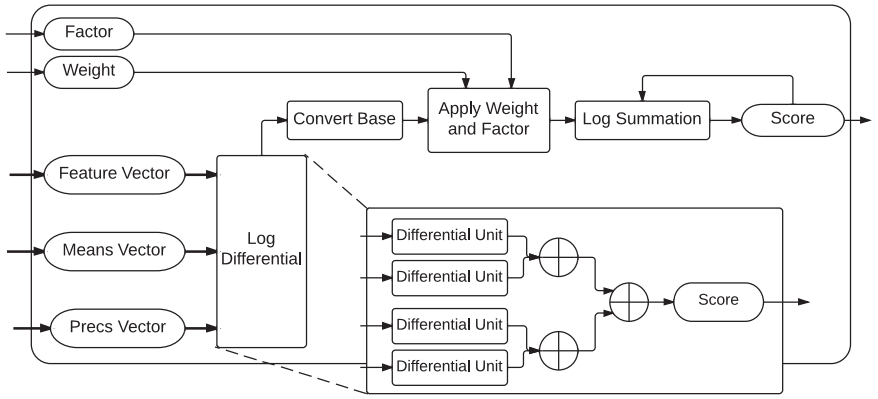


Fig. 11. FPGA GMM diagram.

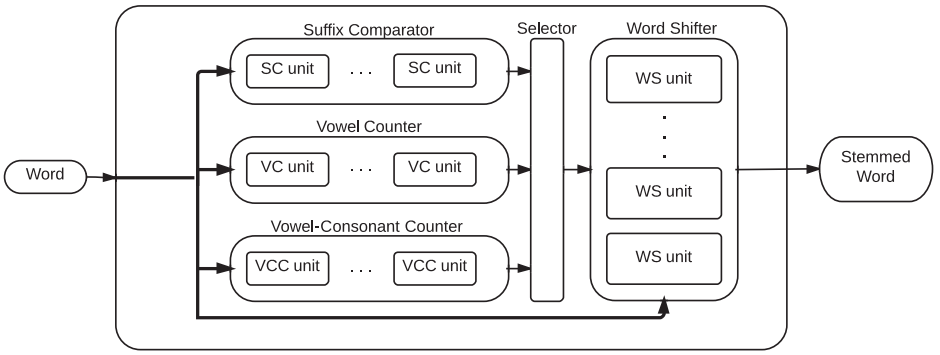


Fig. 12. FPGA stemmer diagram.

Table V. Speedup of Sirius Suite Across Platforms

Service	Benchmark	CMP	GPU	Phi	FPGA
ASR	GMM	3.5	70.0	1.1	169.0
	DNN	6.0*	54.7*	11.2	110.5 [Farabet et al. 2011]
QA	Stemmer	4.0	6.2	5.6	30.0
	Regex	3.9	48.0 [Vasiliadis et al. 2009]	1.1	168.2 [Yang et al. 2008]
	CRF	3.7	3.8 [Piatkowski 2011]	4.7	7.5 [Swaminathan et al. 2002]
IMM	FE	5.2	10.5	2.5	34.6 [Bouris et al. 2010]
	FD	5.9	120.5	12.7	75.5 [Bouris et al. 2010]

*This includes DNN and HMM combined.

achieve a much lower latency than the original Porter algorithm. Our implementation performs multiple vector operations simultaneously to count vowels, vowel-consonant pairs, and compare suffixes. Together, these operations select the correct word shift for the specific step. We formed a single pipelined core based upon six steps dealing with the different possibilities of suffixes. We instantiate multiple cores to fill the FPGA fabric to deliver maximum performance.

4.4. Accelerator Results

Table V and Figure 13 present the performance speedup achieved by the Sirius kernels running on each accelerator platform, organized by service type. For the numbers

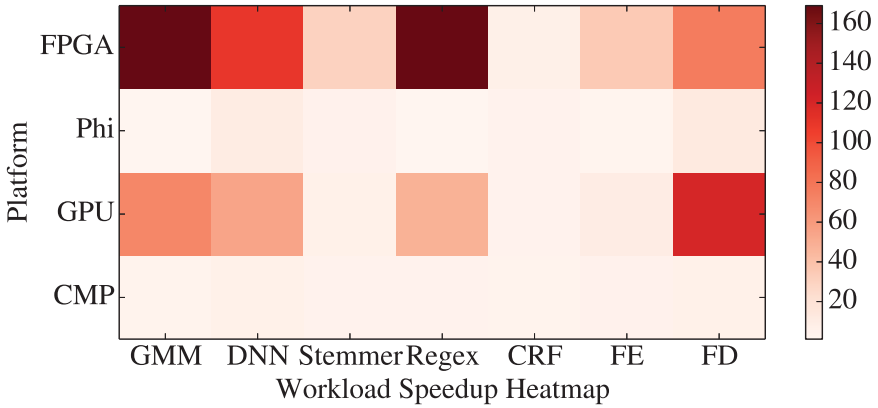


Fig. 13. Heatmap of acceleration results.

from the prior literature, we scale the FPGA speedup number to match our FPGA platform based on fabric usage and area reported in prior work. We also use numbers from the literature for kernels (Regex and CRF) that were already ported to the GPU architecture and yielded better speedups than our implementations.

4.4.1. ASR. The GMM implementation, extracted from CMU Sphinx's acoustic scoring, had the best performance on the GPU (70×) after optimizations. These custom optimizations on the GPU achieved an order of magnitude improvement by optimizing the data structure layout to ensure coalesced global memory accesses. This leveraged concurrent reads to sequential memory positions for a warp of 32 threads. In addition, it was possible to store the entire data required for the GMM in the GPU memory (2GB) during the deployment time, reducing communication between the host and device. The Phi platform did not perform as well as the GPU, indicating that the custom compiler may not have achieved the optimal data layout. The FPGA implementation using a single GMM core achieved a speedup of 56×; when fully utilizing the FPGA fabric, we achieved a 169× speedup using three GMM cores. RWTH's DNN includes both multithreaded and GPU versions out of the box. The RWTH's DNN parallelizes the entire framework (both HMM search and DNN scoring) and achieves good speedup in both cases. In the cases where we use a custom kernel or cite literature, we assume a 3.7× speedup for the HMM [Chong et al. 2011] as a reasonable lower bound.

4.4.2. QA. The NLP algorithms as a whole have very similar performance across platforms because of the nature of the workload: high input variability with many test statements causes high branch divergence. Fine-tuning the stemming algorithm on the Phi to spawn 120 threads instead of the maximum and switching from allocating a range of data per thread to interlaced array accesses yields a better performance given the lower number of threads used. The FPGA stemmer implementation achieved a 6× speedup over the baseline with a single core using only 17% of the FPGA. Scaling the number of cores to fully utilize the resources of the FPGA yielded a 30× speedup over the baseline. The stemmer algorithm contains many test statements and is not well suited for SIMD operations. We attempted to improve our initial stemmer implementation for the GPU by replacing most of the conditional branches with efficient XOR operations [Singh et al. 2010]. However, our fine-grained XOR-based implementation performed worse than our initial version due to additional synchronization between threads.

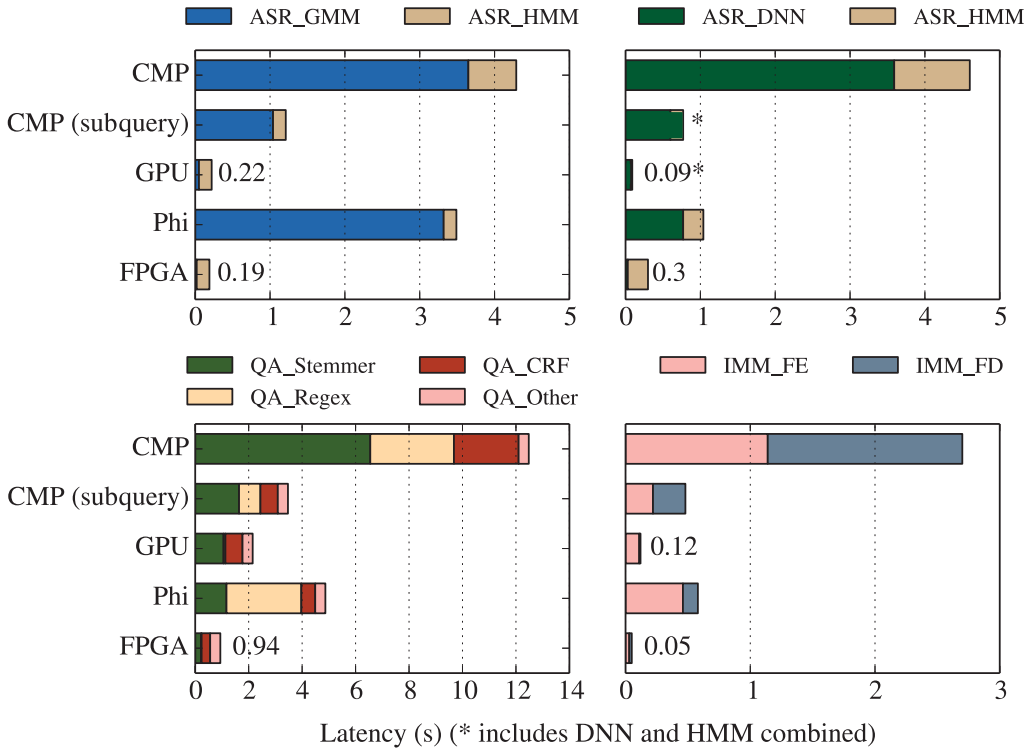


Fig. 14. Latency across platforms for each service.

4.4.3. Image Matching. The image processing kernels achieved the best speedup on the GPU, which uses heavily optimized OpenCV [Bradski 2000] SURF implementations yielding speedups of 10.5× and 120.5× for FE and FD, respectively. Prior work shows that FPGA yields better FE speedups but does not show similar increases for FD. The tiled multicore version yields good speedup, but the performance does not scale as well on the Phi because the number of tiles is fixed, which means that there is little advantage to having more threads available. The GPU version has better performance because it uses a data layout explicitly optimized for a larger number of threads.

5. IMPLICATIONS FOR FUTURE SERVER DESIGN

In this section, we investigate the performance, power, and cost-efficiency trade-offs when configuring servers with different accelerator platforms for Sirius.

5.1. Server Level Design

We first investigate the end-to-end latency reduction and the power efficiency achieved across server configurations for Sirius' services, including ASR, QA, and IMM.

5.1.1. Latency Improvement. Figure 14 presents the end-to-end query latency across Sirius' services on a single leaf node configured with each accelerator. We present both results for ASRs that use GMM/HMM and DNN/HMM as key algorithms. The latency breakdown for all hot components within a service is also presented in the figure. For QA, we focus on the NLP components comprising 88% of the cycles of QA, as search has already been well studied [Ferdman et al. 2012].

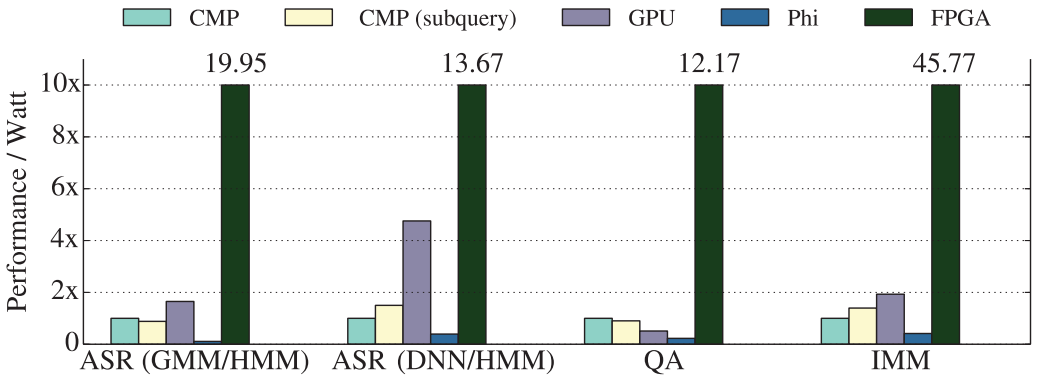


Fig. 15. Performance per watt.

Table VI. Platform Power and Cost

Platform	Power TDP (W)	Cost ($)
Intel Xeon CPU E3-1240	80	250
NVIDIA GPU GTX 770	230	399
Intel Xeon Phi 5110P	225	2,437
Xilinx Virtex-6 FPGA	22	1,795

Our baseline in this figure—CMP—is the latency of the original algorithm implementations of Sirius running on a single core of an Intel Haswell server, described in Table III. CMP (subquery) is our Pthreaded implementation of each service exploiting parallelism within a single query, thus reducing the single query latency. This is executed on four cores (eight hardware threads) of the Intel Haswell server. CMP (subquery) in general achieves a 25% latency reduction over the baseline. Across all services, the GPU and FPGA significantly reduce the query latency. For example, the FPGA implementation of ASR (GMM/HMM) reduces the speech recognition query latency from 4.2s to only 0.19s. The FPGA outperforms the GPU for most of the services except ASR (DNN/HMM). Although Intel Phi can reduce the latency over the single core baseline (CMP), Phi is generally slower than the Pthreaded multicore baseline.

5.1.2. Energy Efficiency. Figure 15 presents the energy efficiency (performance/watt) for each accelerator platform across four services of the Sirius pipeline, normalized by the performance/watt achieved by using all cores on a multicore CPU by query-level parallelism. Here performance is defined as $1/latency$. Table VI presents the power (TDP) for each accelerator platform. The FPGA has the best performance/watt, exceeding every other platform by a significant margin, with more than 12× energy efficiency over the baseline multicore. The GPU's performance/watt is also higher than the baseline for three of four services. Its performance/watt is worse than the baseline for QA, mainly due to its moderate performance improvement for this service.

5.2. DC Design

Based on the latency and energy-efficiency trade-offs for server platforms discussed in the previous section, we evaluate multiple design choices for DCs composed of accelerated servers to improve performance (throughput) and reduce the TCO.

5.2.1. Throughput Improvement. The latency reduction shown in Figure 14 can translate to significant throughput improvement. Figure 16 presents the throughput improvement achieved using various acceleration platforms without degrading latency beyond

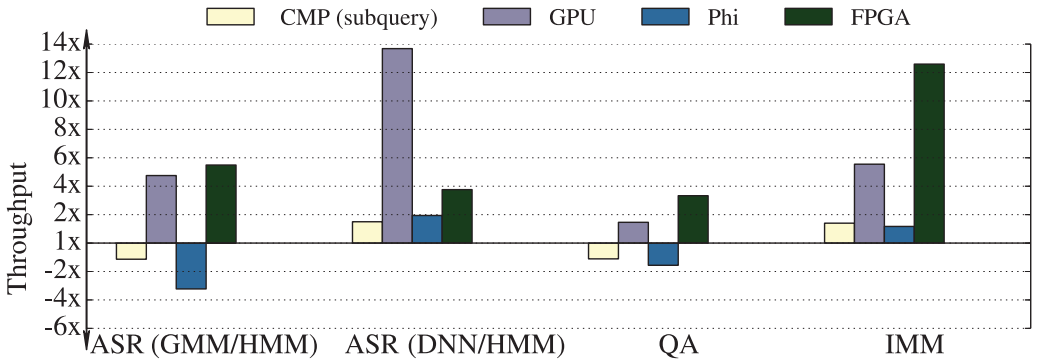


Fig. 16. Throughput across services.

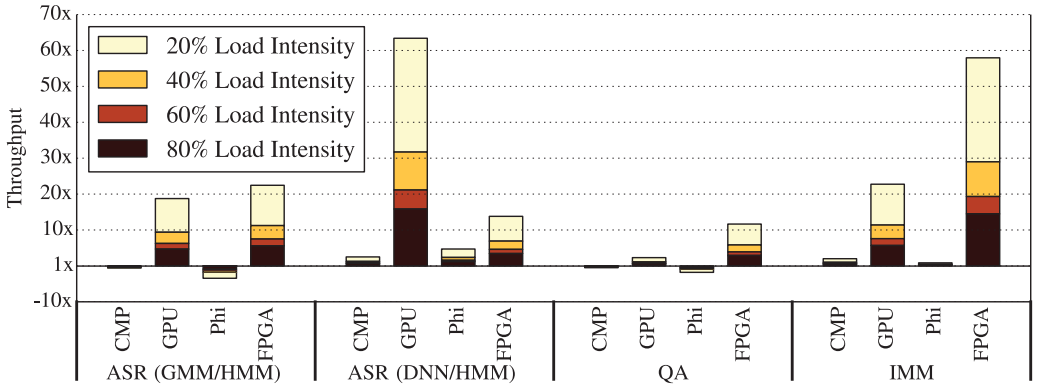


Fig. 17. Throughput improvement at various load levels modeled as an M/M/1 queue (darker is a higher load intensity for each platform).

the baseline. Similar to Figures 14 and 15, the CMP baseline executes the original Sirius workload on the Intel Haswell platform, where all four cores are utilized to serve queries, thus achieving similar throughput as CMP (subquery level). Note, however, that CMP's query latency is significantly longer because CMP (subquery level) exploits parallelism within a single query. Figure 16 demonstrates that significant latency reductions achieved by the GPU and FPGA translate to significant throughput improvement. For example, the GPU provides 13.7× throughput improvement over the baseline CMP for ASR (DNN/HMM), whereas the FPGA achieves 12.6× throughput for IMM. For QA, the throughput improvement across the platforms is generally more limited than other services.

Figure 17 presents the throughput improvement achieved using each acceleration platform at various load levels (the server is modeled as an M/M/1 queue). Compared to Figure 16, which presents the throughput improvement at 100% load, when considering queuing effect, the lower the server load, the bigger impact latency reduction would have on throughput improvement. In other words, Figure 16 demonstrates a lower bound of throughput improvement for a queuing system. Since DC servers often operate at medium-to-low load, as shown in Figure 17, significant higher throughput improvement can be expected.

5.2.2. TCO Analysis. Improving throughput allows us to reduce the amount of computing resources (servers) needed to serve a given load. However, reducing the number of

Table VII. TCO Model Parameters

Parameter	Value
DC Depreciation Time	12 years
Server Depreciation Time	3 years
Average Server Utilization	45%
Electricity Cost	$0.067/kWh
DC Price	$10/W
DC Opex	$0.04/W
Server Opex	5% of Capex/year
Server Price (baseline)	$2,102 [ThinkMate 2014]
Server Power (baseline)	163.6W [ThinkMate 2014]
PUE	1.1

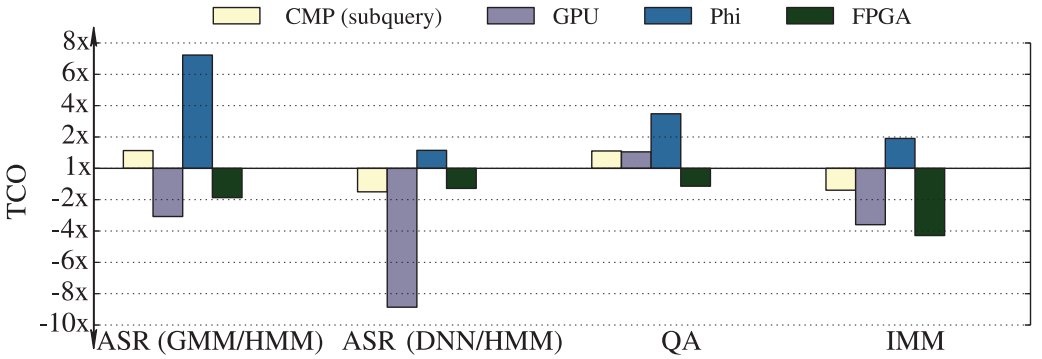


Fig. 18. TCO across platforms for each service.

servers may or may not lead to reduction in the TCO of a DC. Although reducing the machines leads to reduction on DC construction cost and power/cooling infrastructure cost, we may increase the per-server capital or operational expenditure cost either by additional accelerator purchase cost or the energy cost. Here we present a cost analysis to evaluate the implication on the DC cost when using each accelerated server platform.

We perform our TCO analysis using the TCO model recently proposed by Google [Barroso et al. 2013]. The parameters used in our TCO model are described in Table VII. The server price and power usage are based on the following server configuration based on the OpenCompute Project: 1 CPU Intel Xeon E3-1240 V3 3.4GHz, 32GB of RAM, and two 4TB disks [ThinkMate 2014].

Figure 18 presents the DC TCOs with various acceleration options, normalized by the TCO achieved by a DC that uses only CMPs. Overall, the FPGA and GPU provide high TCO reduction. For example, the GPU achieves more than 8× TCO reduction for ASR (DNN) and the FPGA achieves over 4× TCO reduction for IMM. We will further discuss the TCO results and use them to derive our DC designs in the next section.

5.2.3. Homogeneous Datacenter Design. Based on latency results from Figure 14 and TCO results from Figure 18, we first investigate the trade-offs when designing a homogeneous DC—that is, all servers in the DC have the same configuration. Homogeneous DCs are often desirable, as they minimize the management and maintenance overhead [Mars and Tang 2013].

When designing a DC, it would be ideal to maximize performance (e.g., minimize query latency or improve throughput for a given latency constraint) and minimize the TCO. However, trade-offs may need to be made as to which objective should be prioritized if both cannot be optimized by the same design. Figure 19 presents the trade-offs between the query latency improvement and the TCO improvement for each

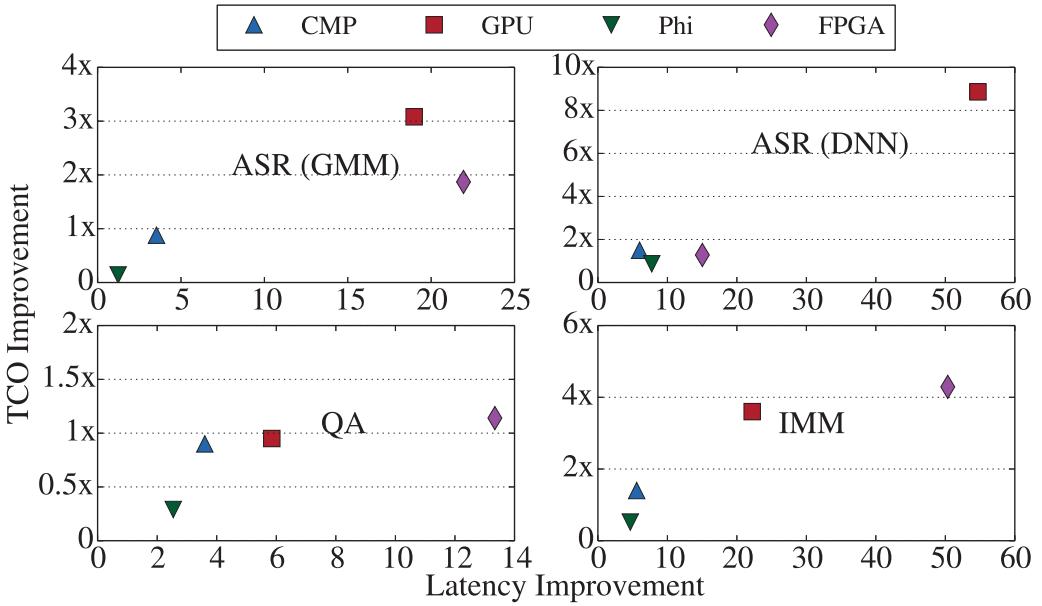


Fig. 19. Trade-off between TCO and latency.

Table VIII. Homogeneous DC

		Hmg. latency	Hmg. TCO (with L constraint)	Without {FPGA, GPU}
With FPGA	ASR (GMM)	FPGA	GPU	FPGA
	ASR (DNN)			
	QA			
	IMM			
Without FPGA	ASR (GMM)	GPU		
	ASR (DNN)			
	QA			
	IMM			
Without {FPGA, GPU}	ASR (GMM)	CMP		
	ASR (DNN)			
	QA			
	IMM			

server option across four Sirius services. The x-axis presents latency improvement, and the y-axis shows the TCO improvement.

As shown in the figure, FPGA achieves the lowest latency (highest latency improvement) among all accelerating platforms for three out of four services that we studied. However, the FPGA's relatively high purchase cost allows GPUs to achieve similar or higher TCO savings as FPGAs with smaller latency reduction. When the FPGA is not considered an option, the GPU achieves the optimal latency and TCO for all services. Even with the FPGA as an accelerator candidate, a GPU-accelerated DC provides the best latency and TCO for ASR using DNN.

Table VIII summarizes the homogeneous DC design for each of the main Sirius services under different conditions and optimization objectives. We present three first-order design objectives: minimizing latency, minimizing TCO with a latency constraint, and maximizing energy efficiency with a latency constraint, shown as three rows of the

Table IX. Heterogeneous DC

		Hetero. Latency	Hetero. TCO (with L constraint)	Hetero. Power Eff. (with L constraint)
With FPGA	ASR (GMM)	FPGA	GPU	FPGA
	ASR (DNN)	GPU(3.6x)		
	QA	FPGA	FPGA(20%)	
	IMM		FPGA(19%)	
Without FPGA	ASR (GMM)	GPU		
	ASR (DNN)			
	QA			
	IMM			
Without {FPGA, GPU}	ASR (GMM)	CMP		
	ASR (DNN)			
	QA			
	IMM			

table. The latency constraint here is CMP (subquery) latency shown in Figure 14. The first row (with FPGA, without FPGA, without FPGA or GPU) also shows the design constraints for the accelerator candidates.

Key observation. In conclusion, FPGAs and GPUs are the top two candidates for homogeneous accelerated DC designs across all three design objectives. An FPGA-accelerated DC allows DCs to minimize latency and maximize energy efficiency for most of the services and is the best homogeneous design option for those objectives. Its power efficiency is desirable for DCs with power constraints, especially for augmenting existing filled DCs that are equipped with capped power infrastructure support. It also improves TCO for all four services. On the other hand, FPGA-accelerated DCs incur higher engineering cost than the rest of the platforms. For DCs where engineering cost needs to be under a certain constraint, GPU-accelerated homogeneous DCs achieve relatively low latency and high throughput. They also achieve similar or higher TCO reduction than FPGA due to the low purchase cost. GPUs could be a desirable option over FPGAs when the high engineering overhead of FGPA implementation is a concern, especially given the quick workload churn (e.g., binaries are updated on the monthly basis) in modern DCs.

5.2.4. Heterogeneous (Partitioned) Datacenter Design. Next we explore the design options for partitioned heterogeneous DCs. Because each service can run on its most suitable platform in a partitioned heterogeneous DC, this strategy may provide additional opportunities for further latency reduction or TCO reduction. Table IX shows various DC design choices for different design objectives (rows), accelerator candidate sets (with FPGA, without FPGA, and without FPGA and GPU), and services (columns). The numbers in parentheses show the improvement on the metric of the specific design objective of that row when the DC design switches from a homogeneous baseline to a heterogeneous partitioned design.

As shown in the first row of the table, when designing a partitioned heterogeneous DC for ASR, QA, and IMM services, if all accelerators are considered viable candidates, GPUs can be used to optimize the latency for ASR (DNN) and achieves 3.6× latency reduction for that service compared to the homogeneous DC using FPGA across all services. Similarly, using FPGAs for QA and IMM achieves 20% and 19% TCO improvement, respectively.

Key observation. In conclusion, the partitioned heterogeneity in our study does not provide much benefit over the homogeneous design. The amount of benefit is certainly

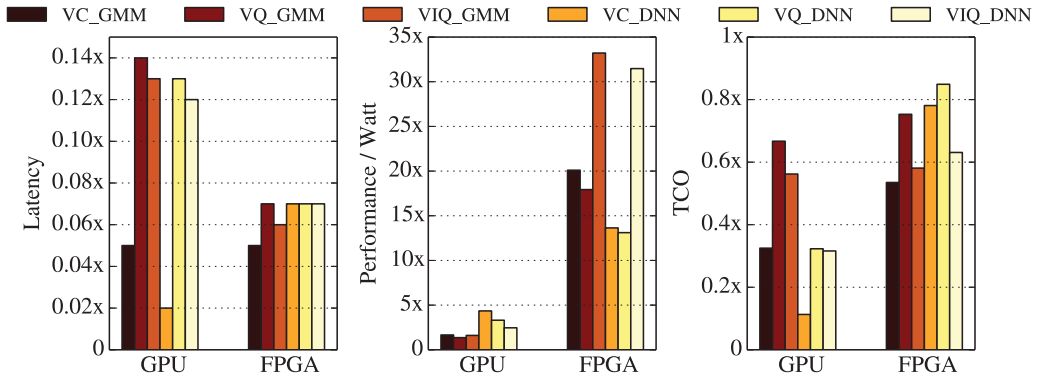


Fig. 20. Latency, energy efficiency, and TCO of GPU and FPGA DCs.

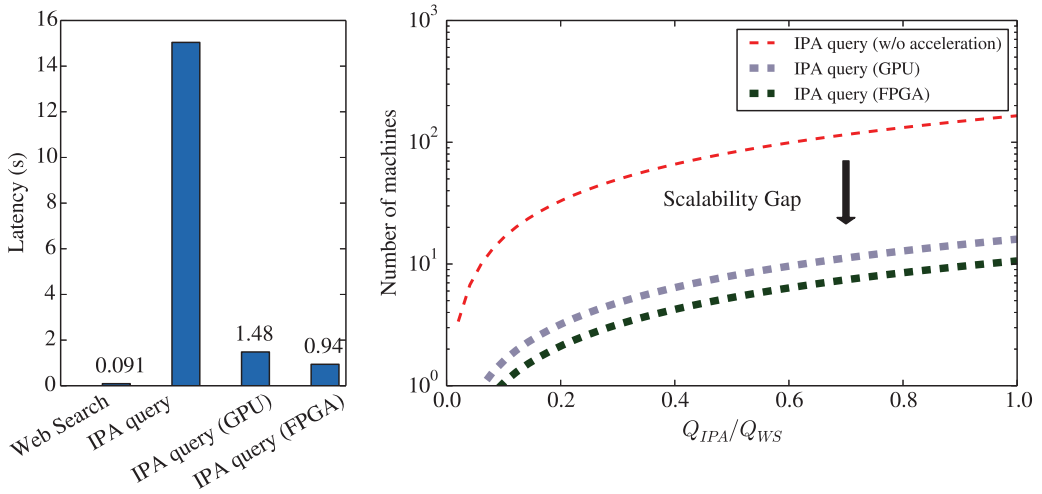


Fig. 21. Bridging the scalability gap.

dependent on the workload partition across services. However, overall, most of the algorithms and services in the Sirius workload exhibit a similar trend in terms of preferences for accelerators for the FPGA and GPU, among others. There is also additional cost associated with managing a heterogeneous/partitioned DC that needs to be justifiable by the performance gain.

5.2.5. Query-Level Results for DC Designs. In previous sections, we focused on latency, energy-efficiency, and TCO trade-offs for various acceleration options across three services in Sirius. In this section, we focus on these trade-offs across three query types supported by Sirius, namely VC, VQ, and VIQ. Figure 20 presents the query latency of three query types achieved by the best two homogeneous DCs, composed of GPU- and FPGA-accelerated servers, respectively. In addition to query latency, energy efficiency of the servers and the TCO of the DCs to support these query types are also presented. GPU-accelerated homogeneous DCs achieve 10× latency reduction on average, and FPGA-accelerated DCs achieve a 16× reduction. The accelerated DCs also reduce the TCO on average by 2.6× and 1.4×, respectively.

Figure 21 further presents the latency reduction of these two accelerated DCs and how homogeneous accelerated DCs can significantly reduce the scalability gap for DCs,

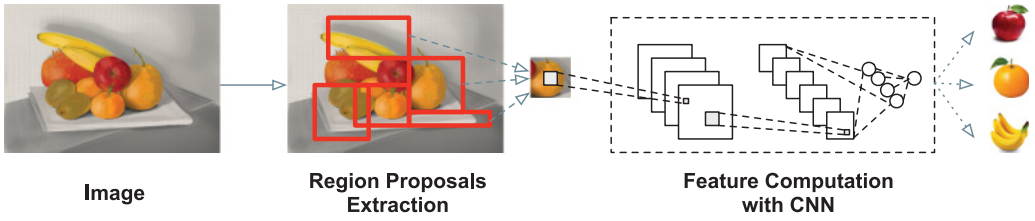


Fig. 22. Object recognition pipeline.

from the current 165× resource scaling, shown in Figure 7, down to 16× and 10× for GPU- and FPGA-accelerated DCs, respectively.

6. EXTENDING SIRIUS WITH OR SERVICE

Recognizing the prevalence of deep learning in WSC applications, we introduce a convolutional neural network (CNN)-based OR service, expanding the capabilities of Sirius with a voice detection query. With this service, we investigate the scaling of our conclusions to recent deep learning–based applications. In this section, we describe the design of an OR service and investigate its real system behavior. Similar to the other Sirius services, we present the study of its implications for future server and DC designs.

6.1. Design of Object Recognition

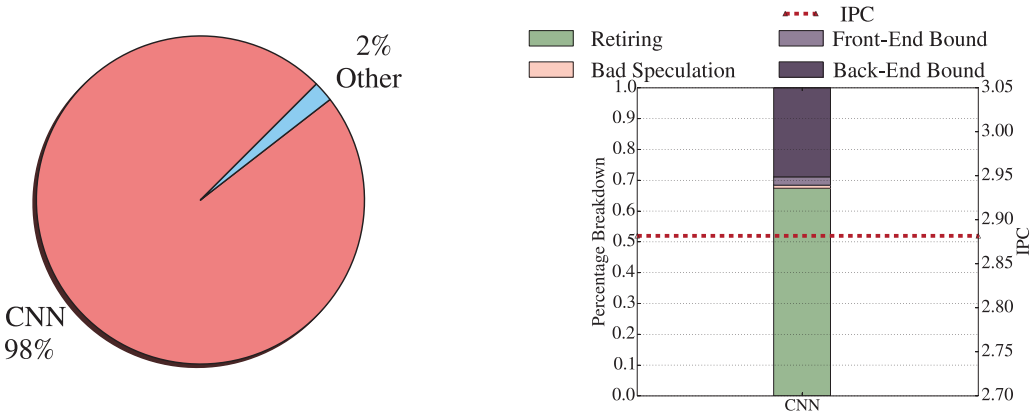
The object recognition design is based on the R-CNN [Girshick et al. 2014] that leverages the strength of CNNs and selective search [Uijlings et al. 2013], which represents the cutting-edge implementation of object recognition in worldwide visual recognition competition [Russakovsky et al. 2015].

Two major components are included in R-CNN [Girshick et al. 2014] to recognize the objects within a image: region proposals extraction and feature computation with CNN, as shown in Figure 22. During region proposals extraction, the image is partitioned and segmented into locations that show high potential in containing the objects to be recognized without exhaustive search. Selective search [Uijlings et al. 2013] is used to exploit the structure of the image as well as apply diverse region grouping strategies to generate a small set of high-quality region proposals. After the region proposals are identified, a large CNN [Krizhevsky et al. 2012] is used to compute the feature vectors for each region proposal. A region proposal is warped to a mean-subtracted 227×227 RGB image and then processed through five convolutional layers and two fully connected layers with forward propagation. At the training time, feature vectors computed by the CNN are used to train a linear classifier for each specific object class, which is then used to recognize the objects within a new image. The dataset used to train the classifier is from the worldwide competition of visual recognition [Russakovsky et al. 2015].

We introduce a new query class named Voice-Detect Query (VDQ) that exercises three Sirius services along its execution pathway: ASR, OR, and QA. A typical VDQ could be a user sending a voice audio asking *Which are the tropical fruits?* along with an image containing several fruits captured via a smart phone or wearable device. We extend our input set to include eight sample queries of this type.

6.2. Real System Analysis of Object Recognition

For our analysis of the OR service, the experiment setup is the same as the rest of the Sirius services (details in Table III). Our real system measurements show that the latency of a VDQ is also orders of magnitude higher than a WS query (25×). Although QA still dominates the query latency, OR consumes 12% of the query execution time.



(a) Cycle Breakdown (b) IPC Breakdown

Fig. 23. Execution breakdown of object recognition.

We also notice that the latency of OR service is quite stable across different queries with variance less than 18ms.

Similar to our study with the rest of Sirius services, we perform top-down profiling to identify the computational bottlenecks of OR service. Figure 23(a) shows that most of the computational cycles (98%) are spent in the CNNs, extracting the features of the image regions and classifying the objects within the regions. Figure 23(b) presents the hardware performance counters profiling the execution of OR service. As the figure shows, OR service executes quite efficiently on Xeon cores and achieves high IPC, leaving little room for performance speedup on the Xeon to address the large scalability gap between WS and VDQ.

6.3. Accelerating Object Recognition and Its Implications

In this section, we accelerate the OR service across four different accelerator platforms and discuss its implications on future server and DC design.

6.3.1. CNN. We extract the key computational bottleneck (CNN) of OR into Sirius Suite. The CNN kernel is then ported to the four accelerator platforms as shown in Table III using the same methodology illustrated in Section 4.3.

Figure 24 presents the performance speedup for object recognition across platforms. The object recognition achieves the best speedup of 31.8× on the FPGA based on the results reported in the literature [Farabet et al. 2011]. We measure the GPU speedup using the cutting edge CNN implementation from NVIDIA cuDNN [2015] and Caffe [Jia et al. 2014] on a GPU, as shown in Table III. The GPU version also improves the latency dramatically with 9.6× speedup. Furthermore, the CMP (subquery) implementation achieves 1.9× speedup, whereas the Phi fails to reduce the latency. This is because CMP takes advantage of batching image segments while the Phi does not.

6.3.2. Implications on Datacenter Design. We evaluate the design choices for DCs in terms of throughput and TCO based on the latency and energy-efficiency results of object recognition on the accelerator platforms.

Using the similar methodology of Figure 16 and 18, we compare the throughput of DCs equipped with these accelerators to a baseline DC composed only of CMPs. As shown in Figure 25(a), the FPGA provides the most throughput improvement by 4.9× over the baseline, whereas the GPU achieves 2.1× throughput improvement.

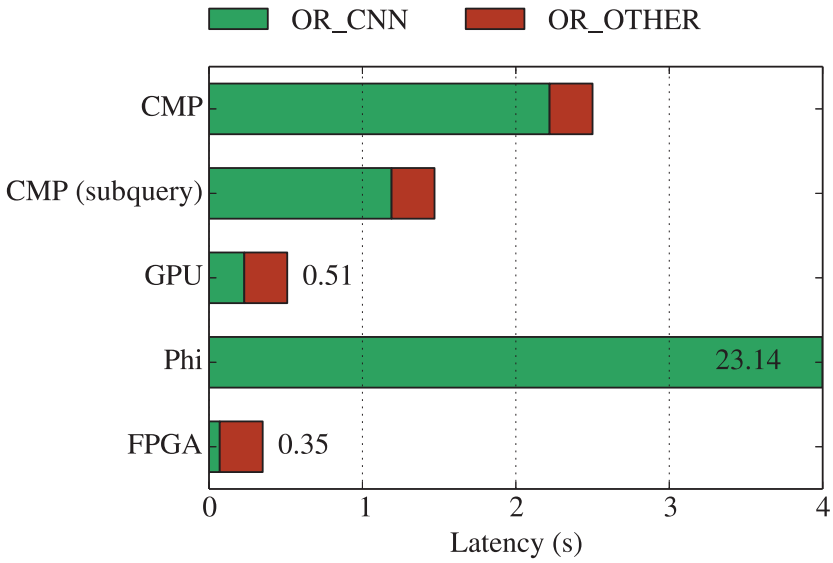


Fig. 24. Latency across platforms.

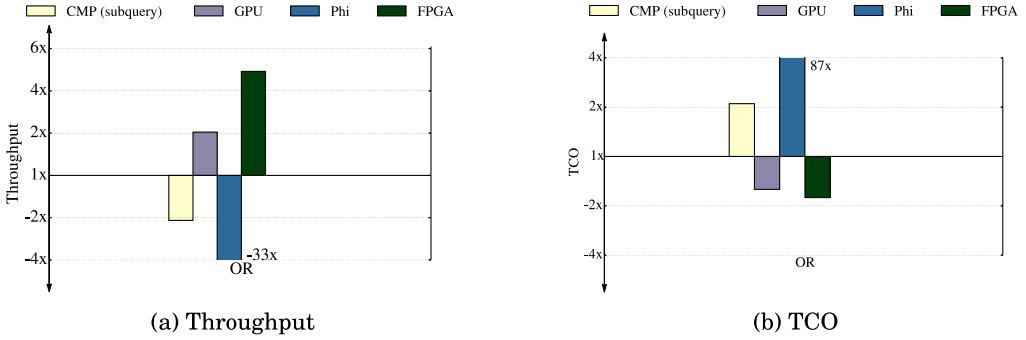


Fig. 25. Throughput and TCO analysis across platforms for OR.

To better understand the implications of latency, energy-efficiency, and throughput improvements on DC design, we provide the TCO analysis results in Figure 25(b) with the TCO model described in Table VII. In general, FPGA and GPU platforms achieve TCO reduction by 1.67× and 1.33×, respectively, over the baseline.

Figure 26 presents the query latency for VDQ type of queries achieved by the best two accelerator platforms using the FPGA and GPU for the DC design. Similar to Figure 20, energy efficiency of the servers and the TCO of the DCs to support this query type are also presented. FPGA-accelerated DCs achieve 14× latency reduction on average, whereas GPU-accelerated DCs achieve 7× latency reduction. The accelerated DCs also improve the energy efficiency on average by 23.4× (FPGA based) and 1.07× (GPU based). In addition, the TCO of the accelerated DCs reduces on average by 1.29× (FPGA based) and 1.94× (GPU based).

Key observation. In conclusion, for object recognition using CNN, the FPGA and GPU platforms are still the best options to provide the most latency reduction (31.8× and 9.6×, respectively) as well as the TCO improvement (1.67× and 1.33×, respectively). The CNN that dominates the computation of object recognition is mainly composed

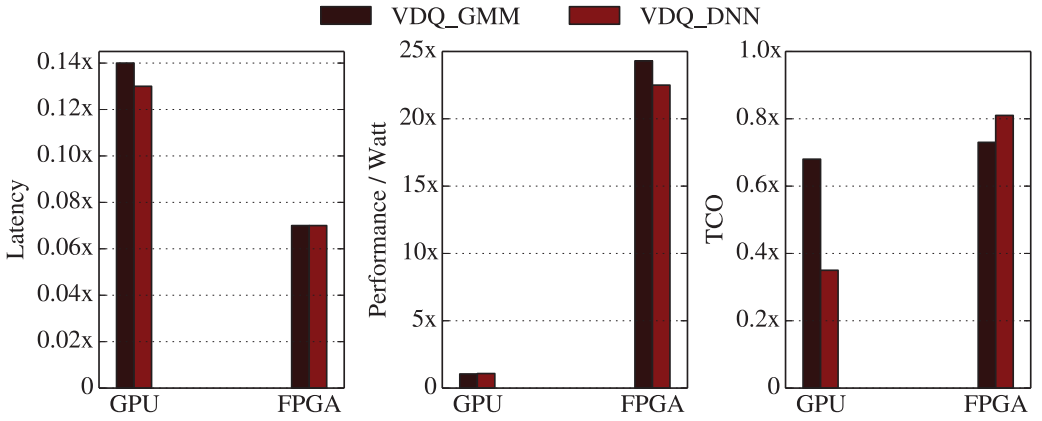


Fig. 26. Latency, energy efficiency, and TCO of GPU and FPGA DCs for VDQ.

of matrix multiplies, which benefits from custom implementations on the GPU and FPGA exploiting the massive parallelism afforded by these accelerators. This translates into significant latency reduction over the general-purpose platform (CMP). Similar to Figure 21, using GPUs and FPGAs to accelerate CNN-based workloads also bridges the scalability gap for future DCs.

7. RELATED WORK

In addition to prior work focusing on DC efficiency [Mars et al. 2011, 2012; Tang et al. 2013a, 2013b; Yang et al. 2013; Mars and Tang 2013; Zhang et al. 2014; Laurenzano et al. 2014; Petrucci et al. 2015; Hsu et al. 2015], a heterogeneous server design [Iyer et al. 2011] was proposed for speech and image recognition, where the GMM scoring and image matching algorithms were ported to hardware accelerators. However, their work does not address the acceleration of NLP algorithms or DNN-based speech recognition. Custom accelerators for specific cloud applications have also been proposed, such as those for memcached [Lim et al. 2013] and database systems [Kocberber et al. 2013] showing the growing need for specialized hardware in server applications. The Catapult project [Putnam et al. 2014] at Microsoft Research has ported key components of Bing's page ranking to FPGAs. In this work, we focus on accelerating the components that make up an IPA, focusing on their impact in the end-to-end system.

Prior work has also investigated acceleration of individual components of Sirius on various platforms. For speech recognition systems using GMM/HMM, prior work characterizes and accelerates the workload in hardware [Krishna et al. 2003; Mathew et al. 2003]. In the past, GPUs were successful in accelerating speech recognition's GMM [Dixon et al. 2009], and more recently ASR was ported using a hybrid CPU-GPU approach [Kim et al. 2012]. The Carnegie Mellon *In Silicon Vox* [Lin et al. 2007] project has implemented an FPGA-based GMM/HMM speech recognizer with a relatively small vocabulary. Image processing algorithms have been shown to map well to accelerators [Hauswald et al. 2014; Rublee et al. 2011; Dinh et al. 2014]. Key NLP techniques also show promising results when ported to hardware [Sun et al. 2014; Lunteren et al. 2012]. Additionally, low-power accelerators for DNNs [Chen et al. 2014; Esmaeilzadeh et al. 2012] have garnered the interest of researchers, as DNNs can be parallelized easily but have better accuracy compared to conventional machine learning techniques [Graves et al. 2013].

8. CONCLUSION

This work introduces Sirius, an open end-to-end IPA application modeled after popular IPA services such as Apple's Siri. Sirius leverages well-established open infrastructures for speech recognition, computer vision, and QA systems. We use Sirius to investigate the performance, power, and cost implications of hardware accelerator–based server architectures for future DC designs. We show that GPU- and FPGA-accelerated servers can improve the query latency on average by 8.5× and 15×. Leveraging the latency reduction, GPU- and FPGA-accelerated servers can reduce the TCO by 2.3× and 1.3×, respectively.

ACKNOWLEDGMENT

We thank our anonymous reviewers for their feedback and suggestions.

REFERENCES

ABIResearch. 2013. Wearable computing devices, like Apple iWatch, will exceed 485 million annual shipments by 2018. Retrieved February 18, 2016, from https://www.abiresearch.com/press/wearable-computing-devices-like-apples-iwatch-will.

ApacheNutch. 2010. Apache Nutch Home Page. Retrieved February 18, 2016, from http://nutch.apache.org.

AppleSiri. 2011. Apple's Siri. Retrieved February 18, 2016, from https://www.apple.com/ios/siri/.

Luiz Andre Barroso, Jimmy Clidaras, and Urs Holzle. 2013. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. Morgan & Claypool.

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. 2006. SURF: Speeded up robust features. In *Computer Vision—ECCV 2006*. Lecture Notes in Computer Science, Vol. 3951. Springer, 404–417.

Dimitris Bouris, Antonis Nikitakis, and Ioannis Papaefstathiou. 2010. Fast and efficient FPGA-based feature detection employing the SURF algorithm. In *Proceedings of the 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'10)*. IEEE, Los Alamitos, CA, 3–10. DOI: http://dx.doi.org/10.1109/FCCM.2010.11

G. Bradski. 2000. *Dr. Dobb's Journal of Software Tools.* OpenCV Library.

Vijay R. Chandrasekhar, David M. Chen, Sam S. Tsai, Ngai-Man Cheung, Huizhong Chen, Gabriel Takacs, Yuriy Reznik, Ramakrishna Vedantham, Radek Grzeszczuk, Jeff Bach, and Bernd Girod. 2011. The Stanford mobile visual search data set. In *Proceedings of the 2nd Annual ACM Conference on Multimedia Systems (MMSys'11)*. ACM, New York, NY, 117–122. DOI: http://dx.doi.org/10.1145/1943552.1943568

Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*. ACM, New York, NY, 269–284. DOI: http://dx.doi.org/10.1145/2541940.2541967

Jike Chong, Ekaterina Gonina, and Kurt Keutzer. 2011. Efficient automatic speech recognition on the GPU. In *GPU Computing Gems Emerald Edition*, W.-M. W. Hwu (Ed.). Morgan Kaufmann, 601–618.

ClarityLab. 2015. Sirius: An Open End-to-End Voice and Vision Personal Assistant. Retrieved February 18, 2016, from http://sirius.clarity-lab.org.

George E. Dahl, Dong Yu, Li Deng, and Alex Acero. 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing* 20, 1, 30–42.

Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS'12)*.

Tung H. Dinh, Dao Q. Vu, Vu-Duc Ngo, Nam Pham Ngoc, and Vu T. Truong. 2014. High throughput FPGA architecture for corner detection in traffic images. In *Proceedings of the 2014 IEEE 5th International Conference on Communications and Electronics (ICCE'14)*. IEEE, Los Alamitos, CA, 297–302.

Paul R. Dixon, Tasuku Oonishi, and Sadaoki Furui. 2009. Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition. *Computer Speech and Language* 23, 4, 510–526. DOI: http://dx.doi.org/10.1016/j.csl.2009.03.005

Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*. IEEE, Los Alamitos, CA, 449–460. DOI: http://dx.doi.org/10.1109/MICRO.2012.48

Clément Farabet, Yann LeCun, Koray Kavukcuoglu, Eugenio Culurciello, Berin Martini, Polina Akselrod, and Selcuk Talay. 2011. Large-scale FPGA-based convolutional networks. In *Scaling Up Machine Learning,* R. Bekkerman, M. Bilenko, and J. Langford (Eds.). Cambridge University Press, 399–419. http://yann.lecun.com/exdb/publis/pdf/farabet-suml-11.pdf.

Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII)*. ACM, New York, NY, 37–48. DOI:http://dx.doi.org/10.1145/2150976.2150982

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. 2010. Building Watson: An overview of the DeepQA project—Ferrucci—AI magazine. *AI MAGAZINE* 31, 3, 59–79. http://www.aaai.org/ojs/index.php/aimagazine/article/view/2303.

G. David Forney Jr. 1973. The Viterbi algorithm. *Proceedings of the IEEE* 61, 3, 268–278.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

GoogleAndroidWear. 2014. Android Wear. Retrieved February 18, 2016, from http://www.android.com/wear/.

GoogleGlass. 2014. Google Glass. Retrieved February 18, 2016, from http://www.google.com/glass.

GoogleNow. 2014. Google Now. Retrieved February 18, 2016, from http://www.google.com/landing/now/.

Alex Graves, Abdel-Rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'13)*. IEEE, Los Alamitos, CA, 6645–6649.

J. Hauswald, T. Manville, Q. Zheng, R. Dreslinski, C. Chakrabarti, and T. Mudge. 2014. A hybrid approach to offloading mobile image classification. In *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'14)*. IEEE, Los Alamitos, CA, 8375–8379.

Marti A. Hearst. 2011. 'Natural' search user interfaces. *Communications of the ACM* 54, 11, 60–67. DOI:http://dx.doi.org/10.1145/2018396.2018414

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel Rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine* Article No. 38131.

Chang-Hong Hsu, Yunqi Zhang, Michael A. Laurenzano, David Meisner, Thomas Wenisch, Lingjia Tang, Jason Mars, and Ron Dreslinski. 2015. Adrenaline: Pinpointing and reigning in tail queries with quick voltage boosting. In *Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, Los Alamitos, CA, 10.

Xuedong Huang, James Baker, and Raj Reddy. 2014. A historical perspective of speech recognition. *Communications of the ACM* 57, 1, 94–103. DOI:http://dx.doi.org/10.1145/2500887

David Huggins-Daines, Mohit Kumar, Arthur Chan, Alan W. Black, Mosur Ravishankar, and Alex I. Rudnicky. 2006. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *Proceedings of the 2006 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 1. IEEE, Los Alamitos, CA, I.

IDCMobile. 2015. Smartphone OS Market Share, 2015 Q2.

IntelVTune. 2015. Intel VTune Home Page. Retrieved February 18, 2016, from https://software.intel.com/en-us/intel-vtune-amplifier-xe.

Ravi Iyer, Sadagopan Srinivasan, Omesh Tickoo, Zhen Fang, Ramesh Illikkal, Steven Zhang, Vineet Chadha, Paul M. Stillwell Jr., and Seung Eun Lee. 2011. CogniServe: Heterogeneous server architecture for large-scale recognition. *IEEE Micro* 31, 3, 20–31.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093.

Jungsuk Kim, Jike Chong, and Ian R. Lane. 2012. Efficient on-the-fly hypothesis rescoring in a hybrid GPU/CPU-based large vocabulary continuous speech recognition engine. In *Proceedings of the 13th Annual Conference on the International Speech Communication Association (INTERSPEECH'12)*.

Onur Kocberber, Boris Grot, Javier Picorel, Babak Falsafi, Kevin Lim, and Parthasarathy Ranganathan. 2013. Meet the walkers: Accelerating index traversals for in-memory databases. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, New York, NY, 468–479.

Rajeev Krishna, Scott Mahlke, and Todd Austin. 2003. Architectural optimizations for low-power, real-time speech recognition. In *Proceedings of the 2003 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'03)*. ACM, New York, NY, 220–231. DOI:http://dx.doi.org/10.1145/951710.951740

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convol utional-neural-networks.pdf.

John Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*. 282–289.

Michael Laurenzano, Yunqi Zhang, Lingjia Tang, and Jason Mars. 2014. Protean code: Achieving near-free online code transformations for warehouse scale computers. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*. ACM, New York, NY.

Kevin Lim, David Meisner, Ali G. Saidi, Parthasarathy Ranganathan, and Thomas F. Wenisch. 2013. Thin servers with smart pipes: Designing SoC accelerators for memcached. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*. ACM, New York, NY, 36–47.

Edward C. Lin, Kai Yu, Rob A. Rutenbar, and Tsuhan Chen. 2007. A 1000-word vocabulary, speaker-independent, continuous live-mode speech recognizer implemented in a single FPGA. In *Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays (FPGA'07)*. ACM, New York, NY, 60–68. DOI:http://dx.doi.org/10.1145/1216919.1216928

Jan Van Lunteren, Christoph Hagleitner, Timothy Heil, Giora Biran, Uzi Shvadron, and Kubilay Atasu. 2012. Designing a programmable wire-speed regular-expression matching accelerator. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*. IEEE, Los Alamitos, CA, 461–472. DOI:http://dx.doi.org/10.1109/MICRO.2012.49

Sergey Lyubka. 2009. SLRE: Super Light Regular Expression Library. Available at http://cesanta.com/.

Jason Mars and Lingjia Tang. 2013. Whare-map: Heterogeneity in homogeneous warehouse-scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*. IEEE, Los Alamitos, CA.

Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*. ACM, New York, NY, 248–259. DOI:http://dx.doi.org/10.1145/2155620.2155650

Jason Mars, Lingjia Tang, Kevin Skadron, Mary Lou Soffa, and Robert Hundt. 2012. Increasing utilization in modern warehouse-scale computers using bubble-up. *IEEE Micro* 32, 3, 88–99. DOI:http://dx.doi.org/10.1109/MM.2012.22

Binu Mathew, Al Davis, and Zhen Fang. 2003. A low-power accelerator for the SPHINX 3 speech recognition system. In *Proceedings of the 2003 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'03)*. ACM, New York, NY, 210–219. DOI:http://dx.doi.org/10.1145/951710.951739

MicrosoftCortana. 2015. Cortana. Retrieved February 18, 2016, from http://www.windowsphone.com/en-us/features-8-1.

MobileMarketing. 2014. Qualcomm Acquires Kooaba Visual Recognition Company. Retrieved February 18, 2016, from http://mobilemarketingmagazine.com/qualcomm-acquires-kooaba-visual-recognition-company/.

NVIDIA cuDNN. 2015. NVIDIA cuDNN: GPU Accelerated Deep Learning. Retrieved February 18, 2016, from https://developer.nvidia.com/cudnn.

Naoaki Okazaki. 2007. CRFsuite: A fast implementation of conditional random fields (CRFs). Retrieved February 18, 2016, from http://www.chokkan.org/software/crfsuite/.

Vinicius Petrucci, Michael A. Laurenzano, Yunqi Zhang, John Doherty, Daniel Mosse, Jason Mars, and Lingjia Tang. 2015. Octopus-man: QoS-driven task management for heterogeneous multicore in warehouse scale computers. In *Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, Los Alamitos, CA, 10.

Nico Piatkowski. 2011. Linear-Chain CRF@GPU. Retrieved February 18, 2016, from http://sfb876.tu-dortmund.de/crfgpu/linear_crf_cuda.html.

Martin F. Porter. 1980. An algorithm for suffix stripping. *Program: Electronic Library and Information Systems* 14, 3, 130–137.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. 2011. The Kaldi speech recognition toolkit. In *Proceedings of the IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE, Los Alamitos, CA.

Andrew Putnam, Adrian Caulfield, Eric Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, Jim Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA'14)*. http://research.microsoft.com/apps/pubs/default.aspx?id=212001.

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the 2011 IEEE International Conference on Computer Vision (ICCV'11)*. IEEE, Los Alamitos, CA, 2564–2571.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3, 211–252. DOI:http://dx.doi.org/10.1007/s11263-015-0816-y

David Rybach, Stefan Hahn, Patrick Lehnen, David Nolden, Martin Sundermeyer, Zoltan Tüske, Siemon Wiesler, Ralf Schlüter, and Hermann Ney. 2011. RASR—the RWTH Aachen University Open Source Speech Recognition Toolkit. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop*.

Frank Seide, Gang Li, and Dong Yu. 2011. Conversational speech transcription using context-dependent deep neural networks. In *Proceedings of the 12th Annual Conference of the International Speech Communication Association (INTERSPEECH'11)*. 437–440. http://msr-waypoint.com/pubs/153169/CD-DNN-HMM-SWB-Interspeech2011-Pub.pdf.

M. G. Siegler. 2011. Apple's Massive New Data Center Set to Host Nuance Tech; Partnership Announcement Due at WWDC. Retrieved February 18, 2016, from http://techcrunch.com/2011/05/09/apple-nuance-data-center-deal/.

A. Singh, N. Kumar, S. Gera, and A. Mittal. 2010. Achieving magnitude order improvement in Porter Stemmer algorithm over multi-core architecture. In *Proceedings of the 2010 7th International Conference on Informatics and Systems (INFOS'10)*. 1–8.

Yuliang Sun, Zilong Wang, Sitao Huang, Lanjun Wang, Yu Wang, Rong Luo, and Huazhong Yang. 2014. Accelerating frequent item counting with FPGA. In *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'14)*. ACM, New York, NY, 109–112. DOI:http://dx.doi.org/10.1145/2554688.2554766

Sriram Swaminathan, Russell Tessier, Dennis Goeckel, and Wayne Burleson. 2002. A dynamically reconfigurable adaptive Viterbi decoder. In *Proceedings of the 2002 ACM/SIGDA 10th International Symposium on Field-Programmable Gate Arrays (FPGA'02)*. ACM, New York, NY, 227–236. DOI:http://dx.doi.org/10.1145/503048.503081

Lingjia Tang, Jason Mars, Wei Wang, Tanima Dey, and Mary Lou Soffa. 2013a. ReQoS: Reactive static/dynamic compilation for QoS in warehouse scale computers. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'13)*. ACM, New York, NY, 89–100. DOI:http://dx.doi.org/10.1145/2451116.2451126

Lingjia Tang, Jason Mars, Xiao Zhang, Robert Hagmann, Robert Hundt, and Eric Tune. 2013b. Optimizing Google's warehouse scale computers: The NUMA experience. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA'13)*. IEEE, Los Alamitos, CA, 188–197. DOI:http://dx.doi.org/10.1109/HPCA.2013.6522318

ThinkMate. 2014. RAX XF2-1130V3-SH. Retrieved February 18, 2016, from http://www.thinkmate.com/system/rax-xf2-1130v3-sh.

Erik F. Tjong, Kim Sang, and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning—Volume 7 (ConLL'00)*. 127–132. DOI:http://dx.doi.org/10.3115/1117601.1117631

Oscar Tackstrom, Dipanjan Das, Slav Petrov, Ryan McDonald, and Joakim Nivre. 2013. Token and type constraints for cross-lingual part-of-speech tagging. *Transactions of the Association for Computational Linguistics* 1, 1–12.

J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. 2013. Selective search for object recognition. *International Journal of Computer Vision* 104, 2, 154–171. https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013.

Giorgos Vasiliadis, Michalis Polychronakis, Spiros Antonatos, Evangelos P. Markatos, and Sotiris Ioannidis. 2009. Regular expression matching on graphics hardware for intrusion detection. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID'09)*. 265–283. DOI: http://dx.doi.org/10.1007/978-3-642-04342-0_14

Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*. ACM, New York, NY, 607–618. DOI: http://dx.doi.org/10.1145/2485922.2485974

Yi-Hua E. Yang, Weirong Jiang, and Viktor K. Prasanna. 2008. Compact architecture for high-throughput regular expression matching on FPGA. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'08)*. ACM, New York, NY, 30–39. DOI: http://dx.doi.org/10.1145/1477942.1477948

Yunqi Zhang, Michael Laurenzano, Jason Mars, and Lingjia Tang. 2014. SMiTe: Precise QoS prediction on real system SMT processors to improve utilization in warehouse scale computers. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*. ACM, New York, NY.

Received October 2015; accepted December 2015