# Sirius Implications for Future Warehouse-Scale Computers

Demand is expected to grow significantly for cloud services that deliver sophisticated artificial intelligence on the critical path of user queries, as is the case with intelligent personal assistants such as Apple's Siri. If the prediction of the trend is correct, these types of applications will likely consume most of the world's computing cycles. The Sirius project was motivated to investigate what this future might look like and how cloud architectures should evolve to achieve it.

Johann Hauswald
Michael A. Laurenzano
Yunqi Zhang
Cheng Li
Austin Rovinski
Arjun Khurana
Ronald G. Dreslinski
Trevor Mudge
University of Michigan

Vinicius Petrucci
Federal University of Bahia

Lingjia Tang
Jason Mars
University of Michigan

"Ultimately, that's why [Clarity Lab] is running [the] Sirius project. The Apples and Googles and … Microsoft know how this new breed of service operates, but the rest of the world doesn't. And they need to."[1]

In this article, we discuss Sirius, an open end-to-end intelligent personal assistant (IPA) application, modeled after popular IPA services such as Apple's Siri. Sirius leverages well-established open infrastructures for speech recognition, image recognition, and question-answering systems. We use Sirius to investigate the performance, power, and cost implications of hardware accelerator-based server architectures for future datacenter designs. Among the popular acceleration options, including GPUs, Intel Phi, and field-programmable gate arrays (FPGAs), the FPGA-accelerated server is the best server option for a homogeneous datacenter design when the design objective is to minimize latency or maximize energy efficiency with a latency constraint. The FPGA achieves an average 16 times reduction on the query latency across various query types over the baseline multicore system. GPUs provide the highest total cost of ownership (TCO) reduction on average. GPU-accelerated servers can achieve an average 10 times query-latency reduction, translating to a 2.6 times TCO reduction (while FPGA-accelerated servers achieve 1.4 times TCO reduction). When excluding FPGAs as an acceleration option, GPUs provide the best latency and cost reduction among the rest of the accelerator choices. Replacing FPGAs with GPUs leads to a 66 percent longer latency, but in return achieves a 47 percent TCO reduction.

## Motivation

Siri, Google's Google Now, and Microsoft's Cortana represent a class of emerging Web service applications known as 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; they recently have



Published by the IEEE Computer Society

been deployed on well-known platforms such as iOS, Android, and Windows Phone, making them ubiquitous on mobile devices worldwide. In addition, the usage scenarios for IPAs are rapidly increasing, with recent offerings in wearable technologies such as smart watches and glasses. Recent projections predict that the wearables market will reach 485 million annual device shipments by 2018.

In this article, we present Sirius, the first open source voice and vision IPA. Prior to the release of Sirius, large companies such as Apple, Google, Microsoft, and Amazon had a monopoly on the intelligent assistant application space. But without access to a representative open intelligent assistant application, researchers cannot investigate the system architecture implications of this type of workload. Sirius is the first effort to serve this purpose. The Sirius study provides insights on the landscape of current accelerator hardware in datacenters for a future in which demand for intelligent assistants grows radically.

Specifically, our study found that IPAs differ from many Web service workloads 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 users with a speech- and/or image-driven contextually based question-and-answer system. Owing to the computational intensity of these components and the large data-driven models they use, service providers house the required computation in massive datacenter platforms in lieu of performing the computation on the mobile devices themselves. This offloading approach is used by both Siri and Google Now as they send compressed recordings of voice commands and queries to datacenters for speech recognition and semantic extraction.[2] However, datacenters have been designed and tuned for traditional Web services, and questions arise as to whether the current design employed by modern datacenters, composed of general-purpose servers, is suitable for emerging IPA workloads.

In particular, IPA queries require a significant amount of computational resources compared to traditional text-based Web services

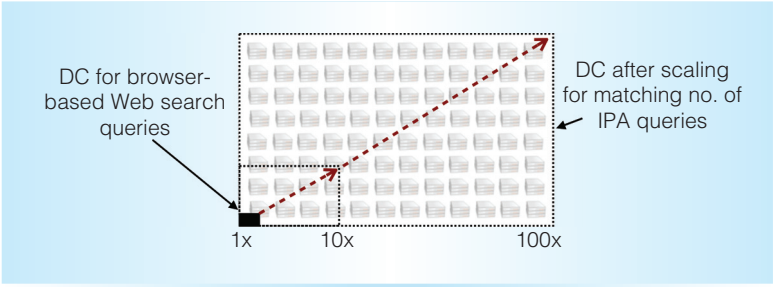


Figure 1. Impact of higher computational requirements for intelligent personal assistant (IPA) queries on datacenters. Illustrated is the scaling of computational resources in a modern datacenter required to sustain an equivalent throughput of IPA queries compared to Web search.

such as search. The computational resources required for a single leaf query, for example, are in excess of 100 times more than that of traditional Web search. Figure 1 illustrates the scaling of computational resources in a modern datacenter required to sustain an equivalent throughput of IPA queries. Because of the looming scalability gap shown in the figure, both academia and industry have expressed significant interest in leveraging hardware acceleration in datacenters using various platforms such as GPUs, many-core coprocessors, and FPGAs to achieve high performance and energy efficiency. To gain further insight on whether there are sufficient acceleration opportunities for IPA workloads and what the best acceleration platform is, we must address the following challenges:

- Identify critical computational and performance bottlenecks throughout the end-to-end lifetime of an IPA query.
- Understand the performance, energy, and cost tradeoffs among popular accelerator options given the characteristics of IPA workloads.
- Design future server and datacenter 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 is prohibitive for investigating the design space of future accelerator-based server designs for this emerging workload. To address this challenge, we constructed Sirius as an end-to-end stand-alone IPA service that implements an

IPA's core functionalities, such as speech recognition, image matching, NLP, and a question-and-answer system. Sirius takes as input user-dictated speech and images captured by a camera. 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.

Sirius also provides the *Sirius Suite,* a benchmark suite composed of the seven computational bottlenecks on a query's pathway that represent 92 percent of that query's execution time. The Sirius Suite also includes implementations of these bottlenecks for a spectrum of computational substrates, including CPU, GPU, FPGAs, and many-core (Phi), all of which are included in the open source release.[3] Sirius was the top-trending open source project on GitHub for the first few weeks after its release. In fact, Sirius and the Sirius Suite have been downloaded tens of thousands of times since their release and are already being used in research papers as well as production projects.

In designing Sirius, we performed investigative research by going to several large companies and talking with key engineers on relevant teams. We asked for insights as to the algorithms and approaches actually used in production, and, using this information, we integrated three services built using well-established open source projects representative of those found in commercial systems. These included Carnegie Mellon University's Sphinx,[4] which represented speech recognition based on the widely used Gaussian mixture model (GMM); Kaldi[5] and RWTH Aachen University's Speech Recognition System (RASR),[6] which represented the industry's recent trend toward speech recognition based on the deep neural network (DNN); OpenEphyra, which represented the-state-of-the-art question-and-answer system based on IBM's Watson[7]; and SURF[8] implemented using OpenCV,[9] which represented the state-of-the-art image-matching algorithms widely used in various production applications. We used these open source projects to steer the construction of both Sirius and the Sirius Suite.

## Sirius: An End-to-End IPA

In this section, we describe the design objectives for Sirius, then present an overview of Sirius and a taxonomy of the query types it supports. Finally, we detail the underlying algorithms and techniques used by Sirius.

### Design Objectives

We had three key objectives when designing Sirius. The first objective was *completeness*: Sirius should provide a complete IPA service that takes the input of human voice and images and provides a response to the user's question with natural language. The second objective was *representativeness*: the computational techniques used by Sirius to provide this response should be representative of state-of-the-art approaches used in commercial domains. The third objective was *deployability*: Sirius should be deployable and fully functional on real systems.

### 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. The audio is processed by an automatic speech recognition (ASR) front end that translates the user's speech question into text. The text then goes through a query classifier that decides whether the speech is an action or a question. For an action, the command is sent back to the mobile device for execution. Otherwise, the Sirius back end receives the question in plain text. The question-answering 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 match the input image to its image database and return relevant information about the matched image using the image-matching service. For example,

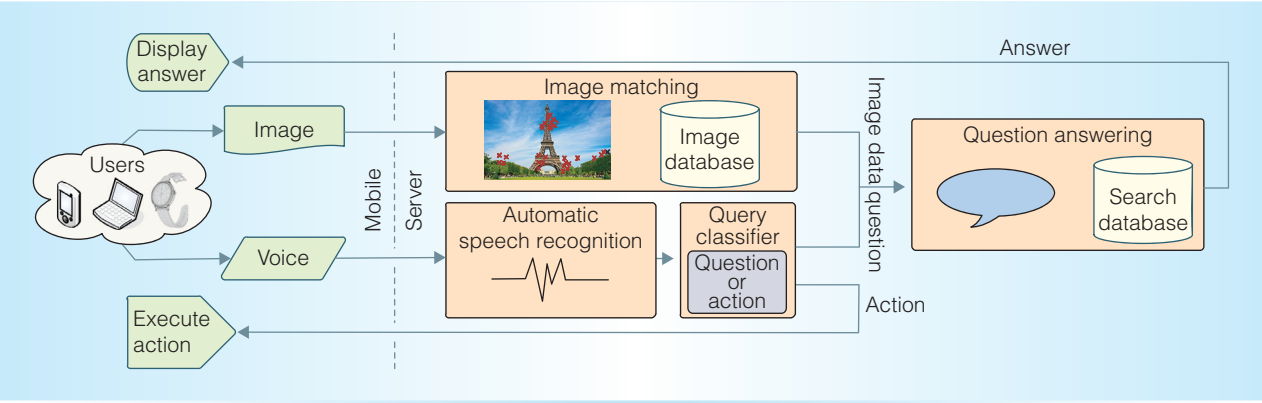


Figure 2. End-to-end diagram of the Sirius pipeline. Sirius is built of three core components that communicate with one another to service IPA queries.

a user can ask, "What time does this restaurant close?" while capturing an image of the restaurant via smart glasses.[3] Sirius can then return an answer to the query based not only on the user's speech but also on information from the image. Table A in "Sirius Web Extra" summarizes the queries supported by Sirius (see http://extras.computer.org/extra/mmi2016030042s1.pdf).

## Design of Sirius: IPA Services and Algorithmic Components

We leverage open infrastructures that use the same algorithms as commercial applications. Speech recognition in Google Voice has used speaker-independent GMMs and hidden Markov models (HMMs) and is adopting DNNs. The OpenEphyra framework used for question answering is an open source release from Carnegie Mellon University's prior research collaboration with IBM on the Watson system.[7] OpenEphyra's NLP techniques, including conditional random field (CRF), have been recognized as state-of-the-art and are used at Google and in other industry question-answering systems.[10] Our image-matching pipeline design is based on the widely used SURF algorithm. We implement SURF using the open source computer vision (OpenCV[9]) library, which is employed in commercial products from companies including Google, IBM, and Microsoft.

## Automatic Speech Recognition Pipeline

The ASR inputs are feature vectors representing the speech segment. The ASR component relies on a combination of an HMM and either a GMM or a DNN. Sirius's GMM-based ASR uses Sphinx,[4] whereas the DNN-based ASR includes Kaldi[5] and RASR.[6]

As Figure 3 shows, 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[11] 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.

## Image-Matching Pipeline

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. Image keypoints are extracted from the input image using the SURF algorithm.[8] In feature extraction, the image is downsampled and convolved multiple times to find interesting keypoints at different scales (see Figure 4). The keypoints are passed to the feature descriptor component, where they are assigned an orientation vector, and similarly oriented keypoints are grouped into feature descriptors. The descriptors from the input image are matched to preclustered descriptors representing the database images

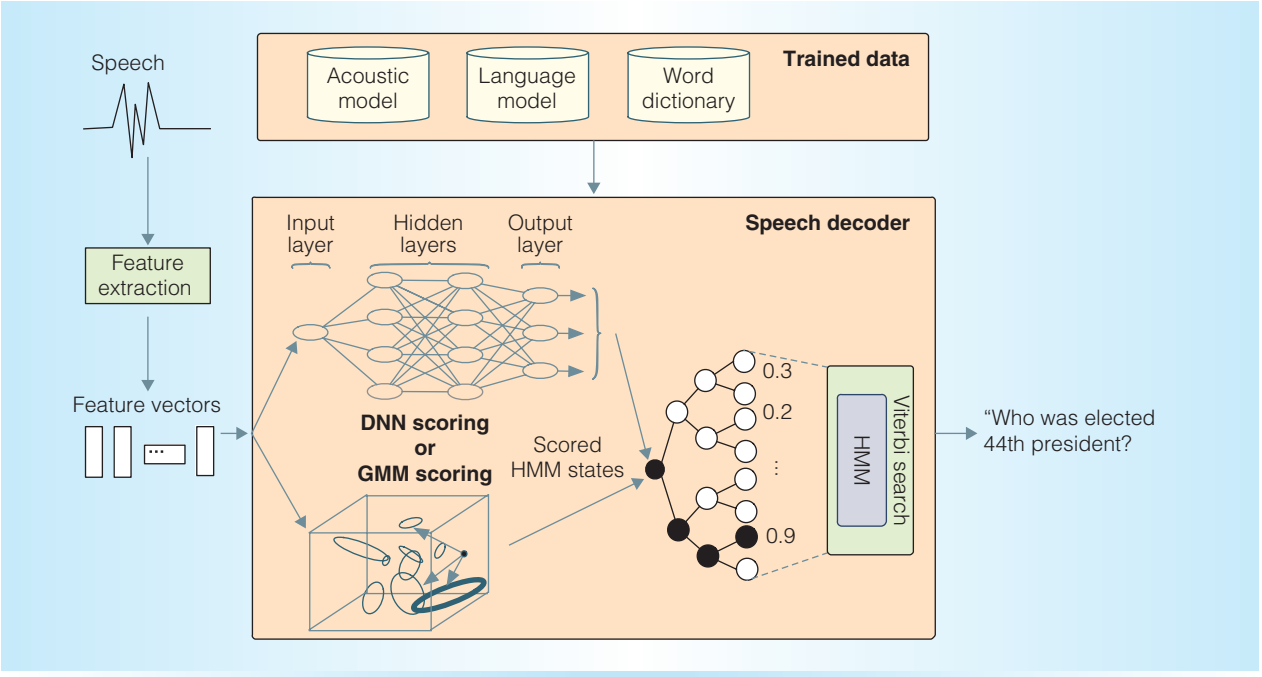


Figure 3. Automatic speech recognition pipeline. The decoding stage receives speech features and uses either a Gaussian mixture model or a deep neural network in combination with a hidden Markov model to transcribe the speech to text.

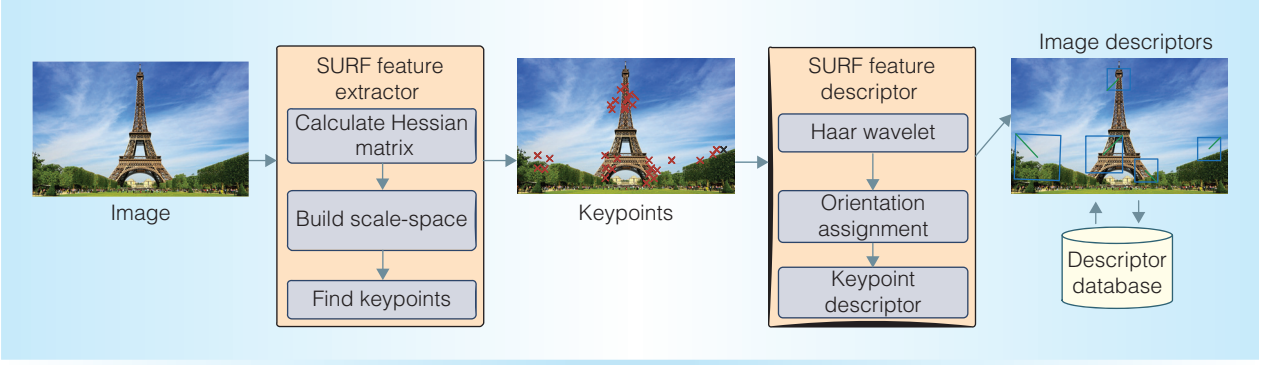


Figure 4. Image-matching pipeline. Keypoints are first extracted from the image and are used to build descriptors (grouping of keypoints). These descriptors are used to find similar images in the database.

using an approximate nearest-neighbor search.

## Question-Answering Pipeline

The text output from the ASR is passed to OpenEphyra,[7] which uses word stemming, regular expression matching, and part-of-speech tagging. Figure 5 shows a diagram of the OpenEphyra engine incorporating these components, generating Web search queries, and filtering the returned results. The Porter Stemming[12] algorithm (stemmer) exposes a word's root by matching and truncating common word endings. OpenEphyra also uses a suite of regular-expression patterns to match common query words. The CRF classifier[2] takes a sentence, the position of each word in the sentence, and the label of the current and previous word as input to make predictions on the part of speech for each word of an input query. Filters using the same techniques extract information from the returned documents; the document with the highest overall score after score aggregation is returned.

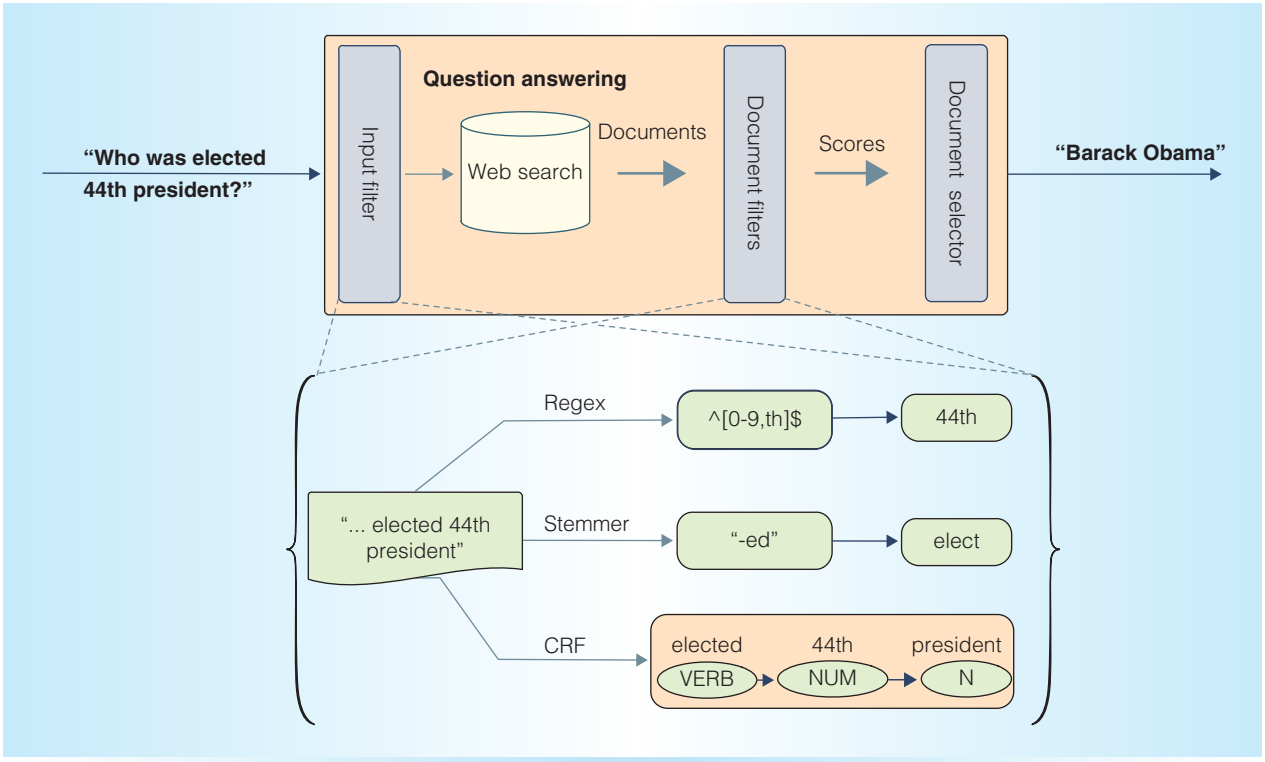


Figure 5. OpenEphyra question-answering pipeline. The QA system uses a combination of natural language processing techniques to generate search queries for the document database and score the returned queries to pick out the best answer.

## 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.

### Scalability Gap

To gain insights on the required resource scaling for IPA queries in modern datacenters, we juxtapose the computational demand of an average Sirius query with that of an average Web search query. We compare the average query latency for both applications on a single core at a very low load.

Figure 6a presents the average latency of both Web search using open source Apache Nutch (http://nutch.apache.org) and Sirius queries. The average Nutch-based Web search query latency is 91 ms on the Haswell-based server. Sirius's query latency is significantly longer, averaging around 15 s across 42 queries spanning our three query classes (voice command, VC; voice query, VQ; and voice image query, VIQ). Based on this significant difference in the computational demand, we perform a back-of-the-envelope calculation of how the computational resources (machines) in current datacenters must scale to match the throughput in queries for IPAs and Web search.

Figure 6b presents the number of machines needed to support IPA queries as the number of queries increases. The *x*-axis shows the ratio between IPA queries and traditional Web search queries. The *y*-axis shows the ratio of computational resources needed to support IPA queries relative to Web search queries. Current datacenter infrastructures will need to scale their computational resources to 165 times their current size when the number of IPA queries scales to match the number of Web search queries. We refer to this throughput difference as the *scalability gap*.

### Cycle Breakdown of Sirius Services

To identify computational bottlenecks, we perform top-down profiling of hot algorithmic components for each service. Figure 7 presents the average cycle breakdown results. We identify the architectural bottlenecks for

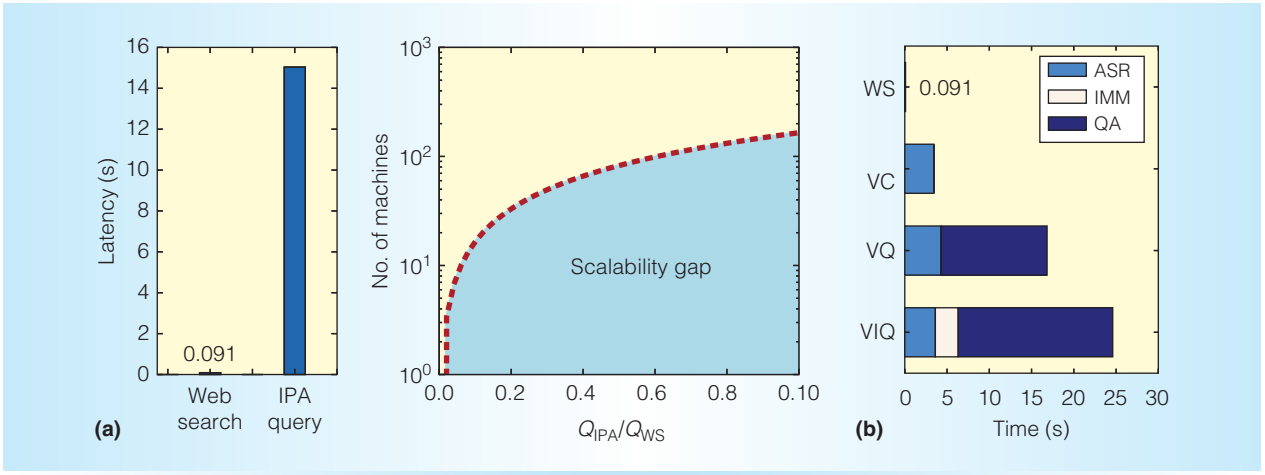


Figure 6. Scalability gap and latency. (a) Average latency of Web search using Apache Nutch and Sirius queries. (b) The number of machines needed to support IPA queries as the number of queries increases. (c) Latency across query types.

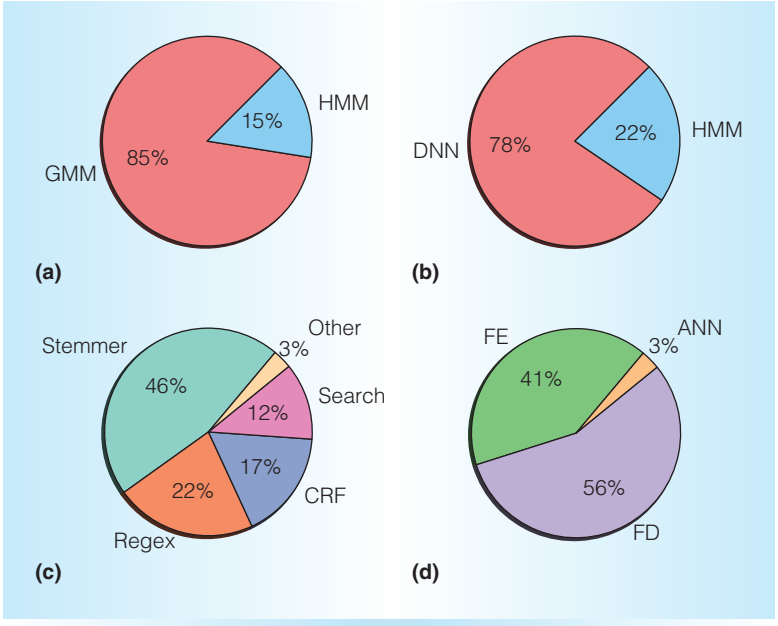


Figure 7. Cycle breakdown per service. (a) ASR (Sphinx); (b) ASR (RASR); (c) QA (OpenEphyra); (d) IMM (SURF). Across the services, several hot components emerge as good candidates for acceleration.

these hot components to investigate the performance improvement potential for a general-purpose processor. From our analysis, even with all stall cycles removed (that is, perfect branch prediction, infinite cache, and so on), the maximum speedup is bound by about three times. Considering the orders of magnitude difference indicated by the scalability gap, further acceleration is needed to bridge the gap.

## 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.

### Accelerator Platforms

We used a total of four platforms, summarized in Table B online, to accelerate Sirius. Our baseline platform was an Intel Xeon Haswell CPU running single-threaded kernels.

Each accelerator platform has advantages and disadvantages. Multicore CPU offers high clock frequency and is not limited by branch divergence, but it has the least amount of threads available. The GPU is massively parallel, but it is also power hungry, requires a custom ISA, has large data transfer overheads, and offers limited branch divergence handling. Intel Phi has a many-core, standard programming model (same ISA); offers optional porting and compiler help; handles branch divergence; and has high bandwidth. On the other hand, it has data transfer overheads and relies on the compiler. Finally, FPGAs can be tailored to implement efficient computation and data layout for the workload, but they also run at a much lower clock frequency, are expensive, and are hard to develop for and maintain with software updates.

**Table 1. The Sirius Suite and granularity of parallelism.**

| Service | Benchmark | Baseline | Input set | Granularity |
|---|---|---|---|---|
| Automatic speech recognition (ASR) | Gaussian mixture model | Sphinx[4] | Hidden Markov model (HMM) states | For each HMM state |
| | Deep neural network | RASR[6] | HMM states | For each matrix multiplication |
| Question answering (QA) | Porter Stemming (stemmer) | Porter[12] | 4M word list | For each individual word |
| | Regular expression (regex) | Super Light Regular Expression Library (SLRE; http://cesanta.com) | 100 expressions/ 400 sentences | For each regex-sentence pair |
| | Conditional random fields (CRFs) | CRFSuite[13] | CoNLL-2000 shared task | For each sentence |
| Image matching (IMM) | Feature extraction (FE) | SURF[8] | JPEG image | For each image tile |
| | Feature description (FD) | SURF[8] | Vector of keypoints | For each keypoint |

## Sirius Suite: A Collection of IPA Computational Bottlenecks

We extracted Sirius's key computational bottlenecks to construct a suite of benchmarks called the Sirius Suite. The Sirius Suite and its implementations across the described accelerator platforms are available alongside the end-to-end Sirius application.[3] We ported 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 Sirius's source code where none were currently available. For each Sirius Suite benchmark, we built an input set representative of IPA queries. The baseline implementations are summarized in column 3 of Table 1. The table also shows the granularity at which each thread performs the computation on the accelerators.

## 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.

## Multicore CPU

We used 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 lets each thread run concurrently and independently, synchronizing only at the end of the execution.

## GPU

We used 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 not currently supported in CUDA for the stemmer kernel.

## Intel Phi

We ported our Pthread versions to the Intel Phi platform, leveraging the target compiler's ability to parallelize the loops on the target platform. To investigate this platform's potential to facilitate ease of programming, we used 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.

## FPGA

We used previously published details of FPGA implementations for several of our Sirius benchmarks in this work. We designed

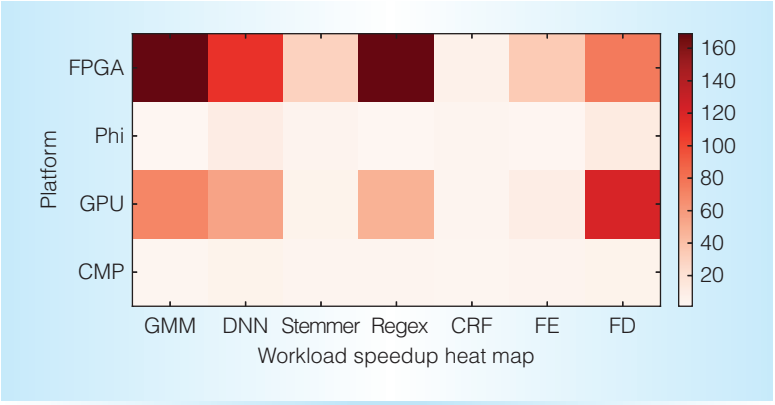


Figure 8. Heat map of acceleration results. The heat map presents the results of accelerating each of the seven kernels (*x*-axis) across the four platforms (*y*-axis), with the darker color representing higher speedups.

our own FPGA implementations for GMM and stemmer and evaluated them on a Xilinx FPGA. Our full paper offers more details on the FPGA designs.[14]

### Accelerator Results

Figure 8 presents the performance speedup achieved by the Sirius kernels running on each accelerator platform. For the numbers from prior literature, we scaled the FPGA speedup number to match our FPGA platform based on the fabric usage and area reported in prior work. We used numbers from the literature for kernels (regex and CRF) that were already ported to the GPU and yielded better speedups than our implementations.

### ASR

The GMM implementation had the best performance on the GPU (70 times) after optimizations. The FPGA implementation using a single GMM core achieved a speedup of 56 times; when fully utilizing the FPGA fabric, we achieved a 169 times speedup using three GMM cores. RWTH's DNN includes both multithreaded and GPU versions out of the box. 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 used a custom kernel or cited literature, we assumed a 3.7 times speedup for the HMM[15] as a reasonable lower bound.

### Question Answering

The NLP algorithms as a whole have similar performance across platforms because of the nature of the workload: high input variability with many test statements causes high branch divergence. The FPGA stemmer implementation achieved 6 times speedup over the baseline with a single core using only 17 percent of the FPGA. Scaling the number of cores to fully utilize the FPGA's resources yielded a 30 times speedup over the baseline.

### Image Matching

The image-processing kernels achieved the best speedup on the GPU that uses heavily optimized OpenCV[9] SURF implementations, yielding speedups of 10.5 and 120.5 times for feature extraction and feature description, respectively. 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 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.

## Implications for Future Server Design

We first investigated the end-to-end latency reduction and power efficiency achieved across server configurations for Sirius's services, including ASR, question answering, and image matching.

### Latency Improvement

Figure 9 presents the end-to-end query latency across Sirius's services on a single leaf node configured with each accelerator. For question answering, we focused on the NLP components comprising 88 percent of the question-answering cycles, because search has already been well studied.

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 B online. CMP (subquery) is our Pthreaded implementation of each service exploiting parallelism within a single query. This is executed on four cores (eight threads) of the Intel Haswell server. CMP (subquery) in general achieves a 25 percent latency reduction over the

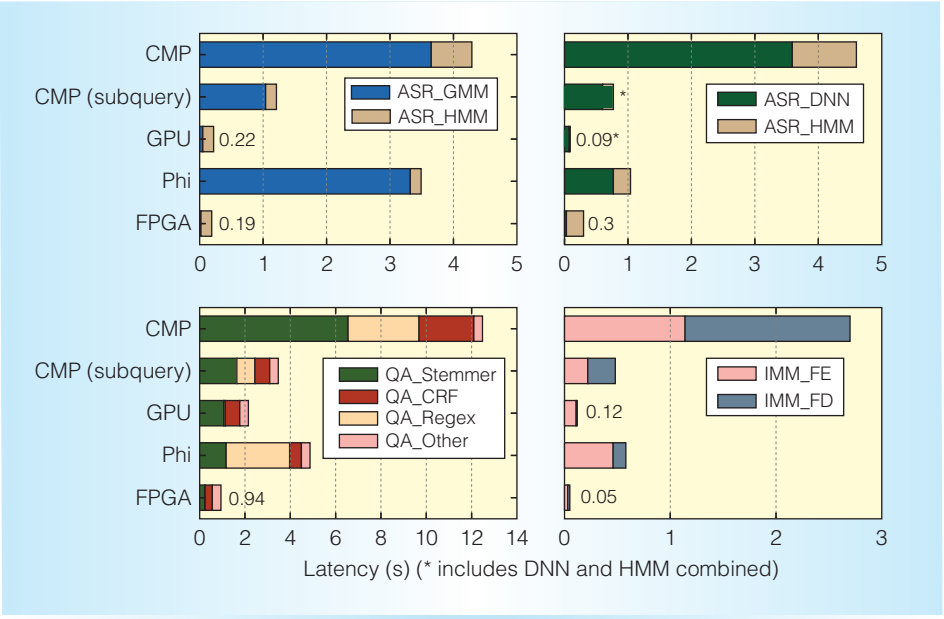


Figure 9. Latency across platforms for each service. The speedups translate into latency gains, with the *x*-axis presenting the end-to-end latency of a single query across the four accelerator platforms.

baseline. Across all services, the GPU and FPGA significantly reduce the query latency.

## Datacenter Design

Next, we evaluate multiple design choices for datacenters composed of accelerated servers to improve performance (throughput) and reduce the TCO. We also investigate each platform's energy efficiency and throughput improvement, which can be found online in the "Energy Efficiency" and "Throughput Improvement" sections of the web extra.

## TCO Analysis

Improving throughput lets us reduce the amount of computing resources (servers) needed to serve a given load. However, this may not necessarily lead to a reduction in a datacenter's TCO. Although reducing the number of machines leads to a reduction in the datacenter construction cost and power/cooling infrastructure cost, we may increase the per-server capital or operational expenditure cost either from the additional accelerator purchase cost or energy cost.

We performed our TCO analysis using the TCO model recently proposed by Google.[16] Table D online gives the parameters used in our TCO model. We based the server price and power usage on the following server configuration based on the OpenCompute Project: 1 CPU Intel Xeon E3 1240 V3 3.4 GHz, 32 Gbytes of RAM, and two 4-Tbyte disks.

Figure 10 presents the datacenter TCO with various accelerator options, normalized by the TCO achieved by a datacenter that uses only CMPs. FPGAs and GPUs provide high TCO reduction. We further discuss the TCO results, derive our datacenter designs, and present query-level results online.

## Overall Latency Reduction Results

Using the derived datacenter design and query-level results, Figure 11 presents the latency reduction of these two accelerated datacenters and how homogeneous accelerated datacenters can significantly reduce the scalability gap, from the current 165 times resource scaling, shown in Figure 6, down to 16 and 10 times for GPU- and FPGA-accelerated datacenters, respectively.

Since the release of Sirius, hundreds of companies around the world have downloaded the Sirius code base. Large companies

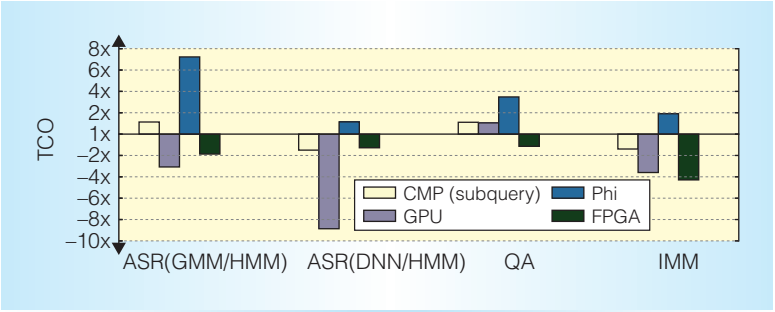


Figure 10. Total cost of ownership across platforms for each service. Bars show the benefits of accelerator DCs normalized to the TCO achieved by a DC that uses only CMPs.

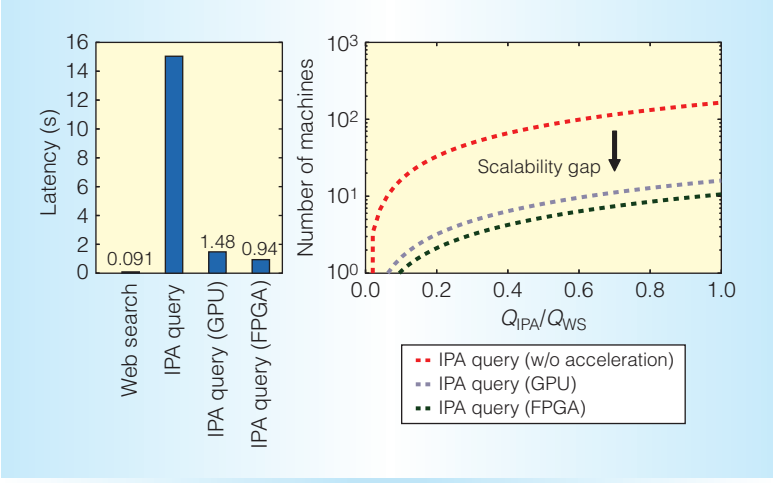


Figure 11. Bridging the scalability gap. The significant latency reductions using GPUs and FPGAs in homogeneous DCs help bridge the scalability gap.

such as Ford, Intel, GE, and Wells Fargo, as well as numerous medium-sized and start-up companies, have demonstrated interest in integrating Sirius into their products. With the release of Sirius, we have made open the type of technology only expected from the big cloud companies. Now any smaller outfit can have a nicely packaged end-to-end solution and a set of tools to design its own IPA. Thus, Sirius represents the democratization of intelligent assistants. Now the technology is in the hands of everyone. MICRO

## References


1. C. Metz, "Voice Control Will Force an Overhaul of the Whole Internet," *Wired*, 24 Mar. 2015; www.wired.com/2015/03/voice-control-will-force-overhaul-whole-internet.
2. J. Lafferty, A. McCallum, and F.C.N. Pereira, *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*, ACM, 2001.
3. "Sirius: An Open End-to-End Voice and Vision Personal Assistant," 2015; http://sirius.clarity-lab.org.
4. D. Huggins-Daines et al., "Pocketsphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices," *Proc. IEEE Int'l Conf. Acoustics, Speech and Signal Processing*, 2006; doi:10.1109/ICASSP.2006.1659988.
5. D. Povey et al., "The Kaldi Speech Recognition Toolkit," *Proc. IEEE Workshop Automatic Speech Recognition and Understanding*, 2011; http://infoscience.epfl.ch/record/192584.
6. D. Rybach et al., "RASR—The RWTH Aachen University Open Source Speech Recognition Toolkit," *Proc. IEEE Automatic Speech Recognition and Understanding Workshop*, 2011.
7. D. Ferrucci et al., "Building Watson: An Overview of the DeepQA Project," *AI Magazine*, vol. 31, no. 3, 2010, pp. 59–79.
8. H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded Up Robust Features," *Computer Vision–ECCV* 2006, Springer, 2006, pp. 404–417.
9. G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, 2008.
10. J. Dean et al., "Large Scale Distributed Deep Networks," *Proc. 25th Conf. Advances in Neural Information Processing Systems*, 2012, pp. 1232–1240.
11. G. David Forney Jr., "The Viterbi Algorithm," *Proc. IEEE*, vol. 61, no. 3, 1973, pp. 268–278.
12. M.F. Porter, "An Algorithm for Suffix Stripping," *Program: Electronic Library and Information Systems*, vol. 14, no. 3, 1980, pp. 130–137.
13. N. Okazaki, "CRFSuite: A Fast Implementation of Conditional Random Fields (CRFs)," blog, 2007; www.chokkan.org/software/crfsuite.

14. J. Hauswald et al., "Sirius: An Open End-to-End Voice and Vision Personal Assistant and its Implications for Future Warehouse Scale Computers," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2015, pp. 223–238.
15. J. Chong, E. Gonina, and K. Keutzer, "Efficient Automatic Speech Recognition on the GPU," *GPU Computing Gems Emerald Edition*, Morgan Kaufmann, 2011.
16. L.A. Barroso, J. Clidaras, and U. Holzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 2nd ed., 2013.

**Johann Hauswald** is a PhD candidate in the Computer Science and Engineering Department at the University of Michigan. He received an MS in computer science and engineering from the University of Michigan. Contact him at jahausw@umich.edu.

**Michael A. Laurenzano** is a PhD candidate in the Computer Science and Engineering Department at the University of Michigan. He received an MS in computer science and engineering from the University of California, San Diego. Contact him at mlaurenz@umich.edu.

**Yunqi Zhang** is a PhD candidate in the Computer Science and Engineering Department at the University of Michigan. He received an MS in computer science and engineering from the University of California, San Diego. Contact him at yunqi@umich.edu.

**Cheng Li** is a PhD student in the Computer Science and Engineering Department at the University of Illinois, Urbana–Champaign. She received an MS in computer science and engineering from the University of Michigan, where she completed the work for this article. Contact her at elfchris@umich.edu.

**Austin Rovinski** is an undergraduate student in electrical engineering at the University of Michigan. Contact him at rovinski@umich.edu.

**Arjun Khurana** is a master's student in the Computer Science and Engineering Department at the University of Michigan. He received a BSE in electrical engineering from the University of Michigan. Contact him at khuranaa@umich.edu.

**Ronald G. Dreslinski** is an assistant professor in the Computer Science and Engineering Department at the University of Michigan. He received a PhD in computer science and engineering from the University of Michigan. Contact him at rdreslin@umich.edu.

**Trevor Mudge** is the Bredt Family Professor of Computer Science and Engineering at the University of Michigan, Ann Arbor. He received a PhD in computer science from the University of Illinois, Urbana–Champaign. Contact him at tnm@umich.edu.

**Vinicius Petrucci** is an assistant professor in the Department of Computer Science at the Federal University of Bahia, Brazil. He received a PhD in computer science from the Fluminense Federal University, Brazil. Contact him at petrucci@dcc.ufba.br.

**Lingjia Tang** is an assistant professor in the Computer Science and Engineering Department at the University of Michigan. She received a PhD in computer science from the University of Virginia. Contact her at lingjia@umich.edu.

**Jason Mars** is an assistant professor in the Computer Science and Engineering Department at the University of Michigan. He received a PhD in computer science from the University of Virginia. Contact him at profmars@umich.edu.