

---

# COGNISERVE: HETEROGENEOUS SERVER ARCHITECTURE FOR LARGE-SCALE RECOGNITION

---

**Ravi Iyer**  
**Sadagopan Srinivasan**  
**Omesh Tickoo**  
**Zhen Fang**  
**Ramesh Illikkal**  
**Steven Zhang**  
**Vineet Chadha**  
**Paul M. Stillwell Jr.**  
Intel Labs  
**Seung Eun Lee**  
Seoul National University  
of Science and  
Technology

AS SMART MOBILE DEVICES BECOME PERVASIVE, VENDORS ARE OFFERING RICH FEATURES SUPPORTED BY CLOUD-BASED SERVERS TO ENHANCE THE USER EXPERIENCE. SUCH SERVERS IMPLEMENT LARGE-SCALE COMPUTING ENVIRONMENTS, WHERE TARGET DATA IS COMPARED TO A MASSIVE PRELOADED DATABASE. COGNISERVE IS A HIGHLY EFFICIENT RECOGNITION SERVER FOR LARGE-SCALE RECOGNITION THAT EMPLOYS A HETEROGENEOUS ARCHITECTURE TO PROVIDE LOW-POWER, HIGH-THROUGHPUT CORES, ALONG WITH APPLICATION-SPECIFIC ACCELERATORS.

.....Over the past decade, Web search has become an important large-scale workload for cloud servers, making it necessary for architects to rethink server architectures from power, cost, and performance perspectives.<sup>1</sup> In the next decade, we expect recognition (of images, speech, gestures, and text) to engender the new wave of features supported on smart mobile devices. For example, mobile augmented reality (MAR) is an upcoming application that lets users point their handheld cameras to an object and, with the help of a cloud server, recognize that object and overlay relevant metadata on it.<sup>2</sup> Similarly, along with the widespread use of handheld devices, there has been a resurgence of interest in enabling users to initiate searches and dictation by speaking as opposed to typing. This also requires the device to work with the cloud server to perform speech recognition efficiently and in real time. Typical operation

at the server involves extracting the most relevant information from target data (image, voice, and text) and mathematically comparing it to a very large database of similar information precomputed from possible matches. The mathematical operations involved can scale to very large databases, depending on the range of inputs supported and the accuracy required. However, providing such recognition services at a large scale on cloud servers requires understanding these workloads' characteristics and designing a server architecture that is highly efficient in terms of power, throughput, and response time.

In this article, we describe the key characteristics of two open-source recognition workloads: image recognition based on Speeded-Up Robust Features (SURF)<sup>3</sup> and OpenCV, and speech recognition based on Sphinx (<http://sourceforge.net/projects/cmuspinx/files>). On the basis of these characteristics, we explore a recognition server

design (CogniServe) that employs a heterogeneous architecture with the following key features:

- several small cores for low power and high throughput,
- application-specific recognition accelerators to improve response latency and energy efficiency, and
- architectural support for general-purpose programming and efficient communication between cores and accelerators.

Our small cores are based on Intel's Atom core, which is already a very low-power core. To demonstrate acceleration of some of the key hot spots in image and speech recognition, we designed and implemented three hardware accelerators: Gaussian mixture model (GMM), match, and interest point detection (IPD).

From a programming-model and communications perspective, accelerators have traditionally been treated as devices in the platform. The programming model is typically based on a clear demarcation of general-purpose core functionality versus special-purpose device functionality. General-purpose user applications have relied on mechanisms such as system calls to isolate the user space processes from directly controlling the hardware. This model is well-suited for I/O devices and coarse-grained accelerators with execution times far higher than the interface overheads. However, for efficient offloading of fine-grained accelerators in heterogeneous recognition servers, a more efficient, intuitive programming interface is needed for accelerators.

Our proposed CogniServe programming model allows for offloading functions on accelerators without incurring significant overhead in system calls and data copies. The architectural support we provide lets cores and accelerators operate in the same virtual-memory domain. We also enable a direct user interface to accelerators via new instructions. By introducing this architectural support, we can provide a general-purpose programming model for accelerators. Moreover, reducing overhead allows offloading at a far lower granularity than

has been possible thus far. In this article, we restrict our description of the programming model to such fine-grained accelerators. However, it's possible to use this same architecture with different accelerators for different target segments in large-scale computing.

## Recognition workloads and characteristics

As users interact more with devices (smartphones, smart displays or TVs, and so on), it's possible to significantly enhance their personalized experience by enabling more natural modes of input and output than with a traditional keyboard and mouse, and by enabling new usage models to provide information customized to the user context. Recognition applications are emerging in both of these areas:

- Various applications recognize speech and gestures as a new, more natural mode of input and output. For example, some applications use speech recognition to make voice calls, dictate notes, or transcribe data. Others use gesture recognition to replace remote controls for smart TVs. Still others are used to enable interactive gaming.
- Some applications recognize objects and text in images and video to enrich context and experience for users by providing additional information—for example, mobile augmented reality.

In this article, we use two of these emerging recognition applications: object recognition in images and speech recognition. We present execution time and hot-spot characteristics for these workloads, which highlights the value of our recognition server architecture.

### Image recognition characteristics

Object recognition in images can be used in various applications. Here, we focus on MAR—a usage model that's rapidly emerging on smartphones and other mobile devices.<sup>2</sup> Figure 1a shows the basic flow of MAR image recognition, which begins with a query image that the user takes with the camera. The intent is to compare this query image against a set of preexisting images in a database for a

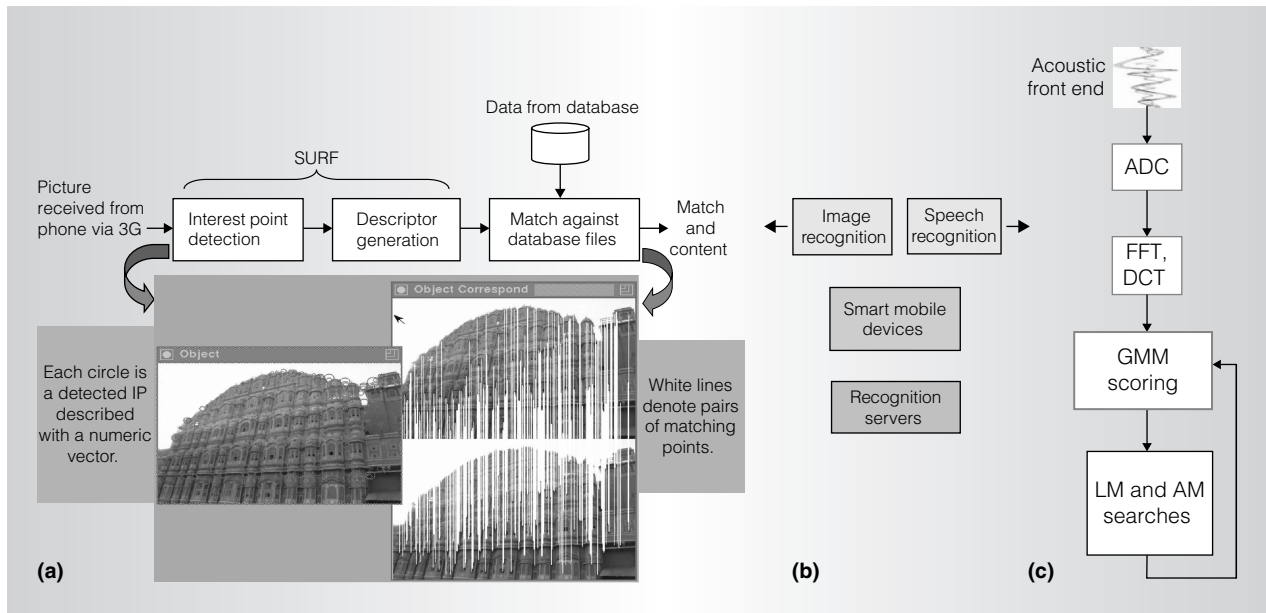


Figure 1. Large-scale recognition for image and speech: image recognition flow detailing various stages, from image capture to database matching (a); recognition architecture employing mobile clients and recognition servers (b); and speech recognition flow detailing various stages for real-time speech identification (c). Thus, we extend the basic recognition architecture to image and speech recognition. (ADC: analog-to-digital converter; AM: acoustic model; DCT: discrete cosine transform; FFT: fast Fourier transform; GMM: Gaussian mixture model; LM: language model; SURF: Speeded-Up Robust Features.)

potential match. This comparison involves three major steps:

1. *Interest-point detection.* Identify interest points in the query image to uniquely characterize the object.
2. *Descriptor generation.* Create descriptor vectors for these identified interest points.
3. *Descriptor matching.* Compare descriptor vectors of the query image against descriptor vectors of stored images in a database (narrowed down by the user context—that is, GPS location or other information).

Researchers have proposed several algorithms to detect interest points and generate descriptors. The most popular are variants of the Scale-Invariant Feature Transform (SIFT)<sup>4</sup> and SURF<sup>3</sup> algorithms. We chose the SURF algorithm for our MAR application because it's faster and sufficiently accurate for the usage model of interest. Researchers have also used SURF successfully in mobile phones for MAR.<sup>2</sup>

Figure 2a shows SURF's execution time when running on two types of platforms: a server platform with traditional large-core microprocessors (Intel's Xeon processor running at 3 GHz) and a netbook platform with small-core microprocessors (Intel's Atom core running at 1.6 GHz). Because we started with open-source code, we also performed significant software optimizations (multithreading, vectorization, and so forth), which Figure 2a labels as "opt." As Figure 2a shows, our software optimizations have already improved image recognition performance by as much as 2×. In addition, the performance difference between large and small cores was roughly 4×. Despite this difference, we explore recognition servers built from small cores, for the following reasons:

- A significant part of this difference comes from the 2× frequency difference between the cores when measured.
- Multiple small cores can fit within a large core's footprint and power envelope.

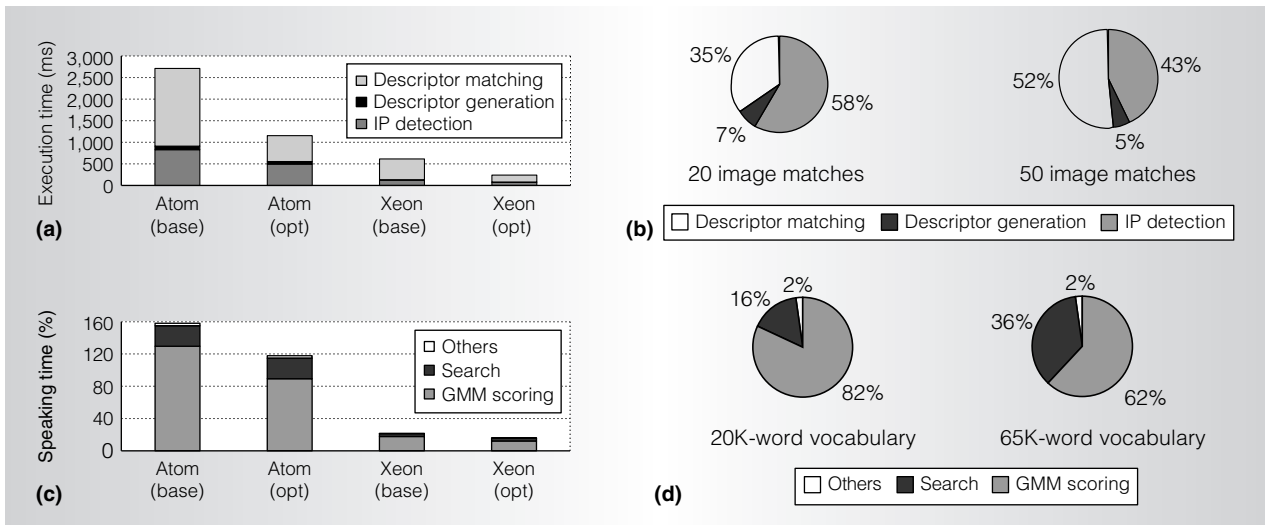


Figure 2. Recognition hot-spot sensitivity: image recognition hot-spot sensitivity on Atom- and Xeon-based platforms (a), image recognition hot-spot sensitivity with database size for database scaling (b), speech recognition hot-spot sensitivity on Atom- and Xeon-based platforms (c), and speech recognition hot-spot sensitivity with vocabulary size for database scaling (d). (opt: optimized.)

- These algorithms' server power consumption and application execution time must be reduced significantly through hardware offloading anyway.

Figure 2b shows the key primitives and hot spots in SURF-based image recognition when running on the Intel Atom core. Descriptor matching takes the most amount of time, followed by IPD (for 50 image matches). The execution time breakdown ratio depends on the number of images to match against. (Extensive workload characteristics and optimizations of image recognition are available elsewhere.<sup>5</sup>)

### Speech recognition characteristics

Automatic speech recognition is particularly attractive as an input method on handheld platforms because of their small form factors. The rapid adoption of mobile devices and the need for enriching context has now made it even more important to enable natural language navigation, accurate transcriptions, and so on. In fact, cloud service providers have already begun enabling a rich user experience based on voice, including voice-based search and other features.

In our work, we target the more difficult speech recognition applications that focus on

large vocabulary, continuous speech recognition (LVCSR). We employ the most widely researched LVCSR software: Carnegie Mellon University's Sphinx 3.0 (<http://sourceforge.net/projects/cmusp3nifiles>). Figure 1c illustrates the basic flow of Sphinx 3, which uses hidden Markov models (HMMs) as statistical models to represent subphonemic speech sounds. There are basically three steps in HMM-based speech recognition systems, including Sphinx 3: the acoustic front end, GMM scoring, and acoustic-model and language-model searches.

Figure 2c shows the execution time of speech recognition on large cores (Intel's Xeon processor) and small cores (Intel's Atom core) when running on speech captured from the *Wall Street Journal* and *HUB-4* broadcast news. Again, we show the impact of software optimizations that we've already performed on the open-source code to speed it up on Intel microprocessors (including multithreading, vectorization, and significant code and algorithmic optimizations). We achieved a speedup of 1.2 $\times$  when applying these optimizations. We also found that the Atom core was about 5 $\times$  to 6 $\times$  slower than the Xeon core when processing speech using the selected open-source engine. As with image recognition, however, we still argue for small cores to be supported in

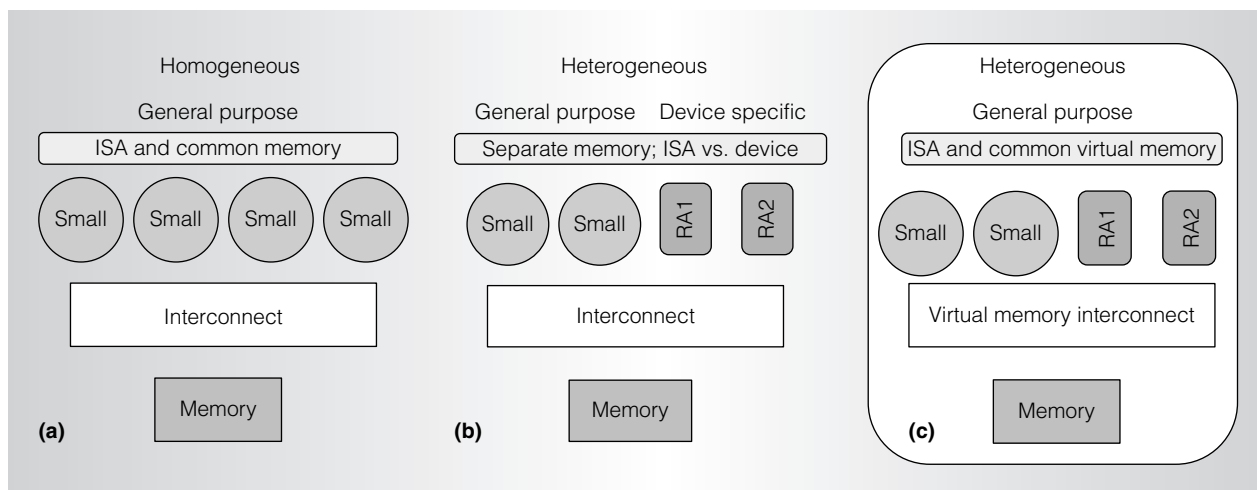


Figure 3. Recognition server: architecture and acceleration on small-core servers (a), driver-based accelerator integration (b), and instruction set architecture (ISA)-based accelerator integration (c). We extended the generic programming model from homogenous architectures to heterogeneous CPU and other device platforms. (RA: recognition accelerator.)

recognition servers, and for the same reasons. In addition, much of this difference comes from large caches (8 Mbytes on the Xeon core versus 512 Kbytes on the Atom).

Figure 2d shows hot-spot profiling of Sphinx 3 on an Atom processor. As the figure shows, GMM scoring takes the most amount of time (62 to 82 percent of processing) followed by HMM (which takes 16 to 36 percent of the total execution time). These were collected with *Wall Street Journal* material with no background noise. Thus, even without noise, the Sphinx 3 decoding time on the Atom is  $1.25\times$  longer than in real time.

In real-world usage cases, environmental noise is unavoidable. Compared with clean audio, decoding time for noisy audio can easily increase by 50 to 300 percent. This shows that hardware acceleration is necessary, not only for power consumption reasons, but also for real-time decoding.

### Recognition server architecture

Figure 3 shows the progression of the recognition server architecture design space that we explored when developing CogniServe. To provide a low-cost, energy-efficient architecture, we developed a recognition server with small cores (like the Intel Atom). Figure 3a shows this basic server architecture, consisting of multiple small cores connected via an

on-die interconnect to an integrated memory controller for attaching it to DRAM. Various commercial and research efforts are already attempting to provide large-scale servers based on Atom cores.<sup>6-8</sup> Our basic design represents a simple homogeneous chip multiprocessor (CMP) architecture, which is a general-purpose CMP from the perspective of programmability ease.

To further accelerate the recognition execution time, we explored the potential of integrating application-specific accelerators for the key recognition primitives. Figure 3b shows this architecture at a high level, in which we replaced some of the small cores with a set of hardware recognition accelerators (RA1, RA2). We included a GMM accelerator for speech recognition because it consumed more than 60 percent of the execution time, a match accelerator for image recognition because it consumed more than 50 percent of the execution time, and an IPD accelerator because it consumed more than 40 percent of the execution time. All three accelerators are key primitives in such algorithms, and therefore are reusable across a range of recognition workloads.

Another important criterion that we used to choose these accelerators is that they primarily involved integer or fixed-point computations. The resulting architecture when integrating accelerators and small cores on

the die is essentially heterogeneous. This architecture provides significant performance improvement and energy efficiency, which is inherent in hardware accelerators as opposed to general-purpose cores. Traditional models essentially treat accelerators like any other I/O device and let developers employ accelerators based on device driver abstraction, as Figure 3b shows.

Unlike the small cores, which employ general-purpose instructions and virtual memory, accelerators are programmed using firmware and device driver interfaces, and they're typically programmed using physical addresses in memory. This makes it difficult to program and also makes the architecture inefficient because it requires control transfer between the user space and the kernel space before accessing the accelerator. Moreover, it requires copying data back and forth between the core and the accelerator for processing.

As Figure 3c shows, to address the lack of general-purpose programming and inefficiency in traditional (device-driver-based) models of heterogeneous architectures, we designed architectural support in the heterogeneous recognition server, enabling direct user access via an instruction set architecture (ISA) from the core to the accelerator (thus eliminating user-to-kernel transition overhead). We also designed common memory management units (MMUs), which let the accelerator share the virtual-memory space with the core so that data movement overheads can be eliminated as well.

## Recognition accelerators in CogniServe

To improve energy efficiency and execution time for a class of special-purpose applications, architects have been exploring the integration of domain-specific hardware accelerators. The most popular hardware accelerator over the past decade has been graphics. However, over the years, researchers have developed application-specific accelerators for video processing, image processing, security and cryptography, and networking. We've designed and implemented the following accelerators for speech and image recognition:

- *GMM accelerator*. This speech recognition accelerator enables GMM scoring

for each of the feature vectors of an audio frame.

- *Match accelerator*. This image recognition accelerator computes distance calculations for matching descriptors from a query image to descriptors from a set of database images.
- *IPD accelerator*. This accelerator identifies the interest points in an input image.

Here, we describe the design of these accelerators in more detail.

### GMM accelerator

Figure 4a shows the GMM accelerator architecture, which includes weighted-square sum calculation of the distances between audio feature vectors and the mean values in the Gaussian table, followed by score generation. The GMM accelerator has a five-stage pipeline, processing 39 different computations concurrently. The arithmetic logic units (ALUs) are 32-bit and 64-bit adders,  $32\text{-bit} \times 16\text{-bit} \rightarrow 48\text{-bit}$  and  $32\text{-bit} \times 32\text{-bit} \rightarrow 64\text{-bit}$  multipliers, and two shifters, all in fixed-point format. The control unit consists of a set of state machines that generate addresses for both fetches from the static RAM (SRAM), and direct memory access (DMA) transfers that copy data from DRAM to SRAM. The DMAs ensure that the audio feature vectors and Gaussian table values are available to the ALUs for the next set of operations.

The Gaussian table for the more than 8,000 senones accounts for most of the memory traffic. (A senone is a set of probability density functions implemented as a Gaussian mixture.) We adopt the audio-frame-grouping technique to save main memory bandwidth consumption. We group four 10-ms audio frames for batch processing, which cuts memory traffic by  $4\times$  at the cost of 40 ms of initial delay.

### Match accelerator

The match accelerator for image recognition computes the Euclidean distance between every pair of descriptor vectors (one interest point from the query image, and one from the database image) and computes the number of matches between two images. Each descriptor has 64 dimensions, represented by 64

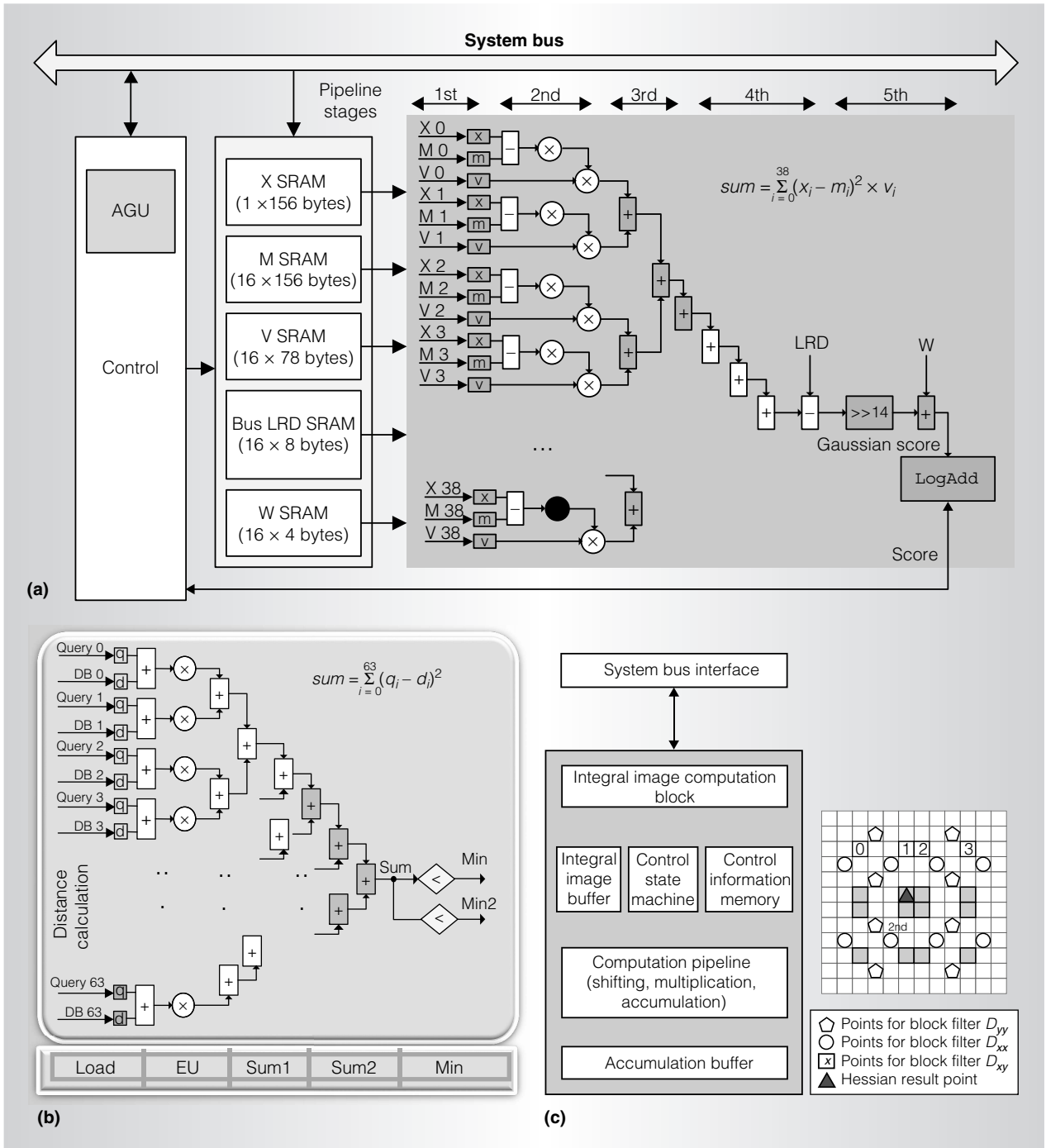


Figure 4. Accelerator designs for recognition hot spots: GMM block for speech recognition (a), mobile augmented reality (MAR) match implementation (b), MAR interest point detection (IPD) accelerator (c). (AGU: address generation unit; DB: database; EU: Euclidian distance square unit; LRD: logarithmic reciprocal determinant; M, X, V, W: input and database vectors; SRAM: static RAM.)

8-bit values (see Figure 4b). The match accelerator employs SRAM as a staging buffer in the address space, without which all subsequent descriptor values would need to be read from DRAM one at a time. Our design uses an SRAM that supports a  $64 \times 64$  descriptor computation (requiring only 8 Kbytes). We rearranged the loop in the

**Table 1. Accelerator performance and speedup from the software implementation.**

Accelerator	Frequency (MHz)	Logic area (mm <sup>2</sup> )	Power (mW)	Speedup
GMM	542	0.63	98.0	6×
Match	568	0.08	23.5	21×
IPD	400	0.36	39.1	13×

brute-force algorithm for 64 descriptor vectors by using a loop-blocking technique. The match accelerator data path requires computing the Euclidean distance square for every pair of descriptor vectors.

The match accelerator has a five-stage pipeline, computing 64 descriptor vectors in 8-bit unsigned integer format concurrently. The load stage loads 64 descriptor vectors from the internal SRAM into the registers. The execution stage calculates the Euclidean distance square between descriptors by computing the difference between a pair of descriptor vector elements and multiplying it by itself to compute the square. The Sum1 and Sum2 stages accumulate 64 Euclidean distance squares. The accumulated value is checked against the minimum and second minimum value in the Min stage, and the results are stored to the internal SRAM for the match operation. The final result, the number of matches between two images, is calculated by accessing the Min and Min2 memory obtained from the Euclidean distance square unit. When the minimum value is less than half the second minimum, the number of matches increases by 1 for the pair of descriptors.

### IPD accelerator

IPD is the first phase in image recognition and comprises steps such as integral image computation and Hessian matrix computation. We designed the IPD accelerator to efficiently implement these steps and provide significant execution time reduction at ultralow power. Figure 4c shows the block filter points for  $D_{xx}$ ,  $D_{yy}$ , and  $D_{xy}$ , which are needed to compute one Hessian matrix result point when an “octave” equals 1 and an “octave layer” equals 1. The IPD accelerator implements the following steps:

1. The accelerator reads a vector of an integral image from the integral image

buffer and sends it to the computation pipeline block.

2. The accelerator reads an entry of the control information table from the control information memory, which includes one associated entry for each block filter point.
3. With the vector from step 1 and the control information from step 2, the computation pipeline block performs shifting, coefficient multiplication, and accumulation. The accumulation buffers store the accumulated results for each block filter.
4. Steps 1 through 3 are repeated until all block filter points are processed. Then, the final Hessian matrix results are calculated and written into either an on-chip buffer or system memory for the next processing step.
5. Steps 1 through 4 are repeated for the entire image.

### Accelerator design evaluation

To evaluate the design of the three accelerators, we conducted extensive register-transfer level (RTL) simulations and integrated each accelerator into a field-programmable gate array (FPGA) platform. We functionally verified that the accelerators operate correctly and produce the correct results. We also synthesized the designs on a 45-nm process and gathered area, frequency, and power statistics.

As Table 1 shows, the accelerators operate at frequencies ranging from 400 MHz to nearly 600 MHz and achieve speedups ranging from 3× to 21× for the kernel. Significant savings in power consumption accompany these speedups. The accelerator logic consumes only about 25 mW to 100 mW in power.

Figures 5a and 5b show the performance gains of image and speech recognition for various hardware configurations and different processor frequency scales and comparison



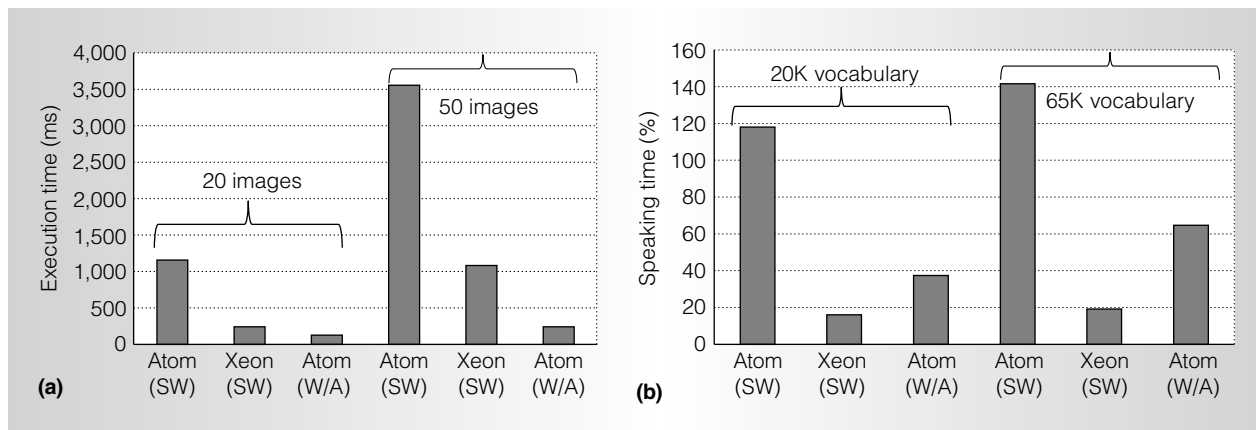


Figure 5. Performance scaling with frequency and database size: performance improvement of image recognition (a) and of speech recognition (b). (SW: software; W/A: with acceleration.)

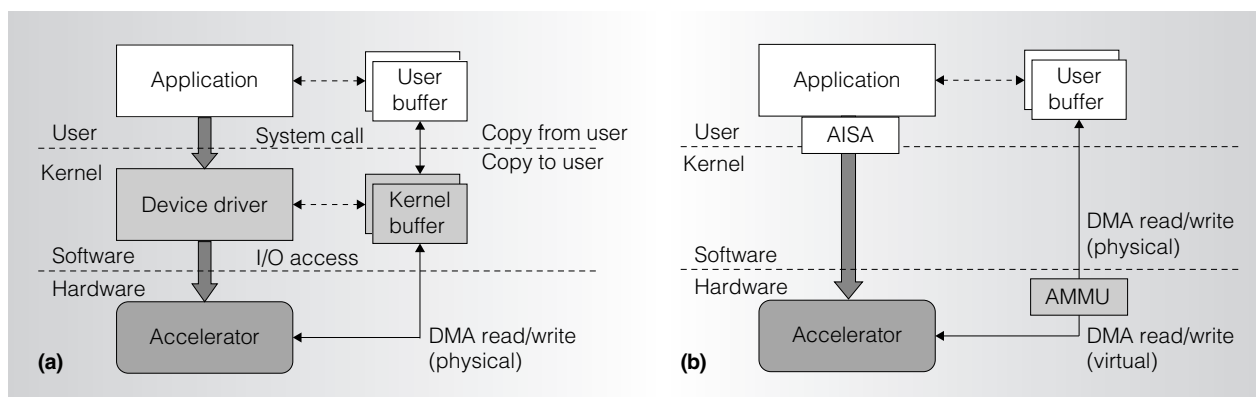


Figure 6. CogniServe accelerator programming and communication: device driver interface (a) and virtual memory-based acceleration (b). (AISA: accelerator ISA; AMMU: accelerator memory management unit; DMA: direct memory access.) The proposed programming model achieves significant overhead reduction by eliminating data copy, buffer allocation, and system-call cycles.

databases. The Atom-based accelerator-integrated platform is about  $2\times$  faster than an Intel Xeon processor for 20 images, and goes up to about  $5\times$  for 50 images. This enables the accelerator-integrated platform to process more queries with lower power consumption. Although our accelerator-integrated platform is about  $2\times$  or  $3\times$  slower than the Intel Xeon for speech recognition, our platform delivers one-third the performance at an order-of-magnitude lower power consumption.

### CogniServe accelerator programming

Traditional interfacing for accelerators has been through device drivers, but this model

poses two challenges: performance loss due to overhead latencies, and lack of a standardized interface for software programming. Extracting the maximum performance and enabling a standardized programming interface requires a new approach to accessing accelerators and communicating with them.

Conventional approaches for accelerator exposure are based on device driver abstraction. Figure 6a shows the overheads with the device driver interfacing model.

- System call overheads range from 100 to 1,000 cycles for each call into the driver (because they require a transition from the user to the kernel).

**Table 2. Performance benefits of the proposed accelerator-programming model.**

Accelerator	Classical driver model (overhead cycles)			Proposed model		
	Data copy	Buffer allocation	System calls and register setup	Total overhead	Total overhead (cycles)	Overhead reduction (%)
MAR IPD	1,110,657	616,355	726	1,727,738	550	99.97
MAR match	33,081	31,757	726	65,565	550	99.16
Sphinx GMM, $N = 8$	3,735	6,378	726	10,840	550	94.93
Sphinx GMM, $N = 1$	3,200	6,096	726	10,022	550	94.51

- Data copy overheads are incurred because the device driver must copy the data from the user to the kernel buffer. The copy overhead is around 2,100 cycles for a 4-Kbyte data transfer. Another approach is to pin the application page and get the physical address using kernel services. Although this approach reduces copy overhead, it adds the cost of address translation.

We address these overheads by introducing the architectural support shown in Figure 6b, which has the performance benefits listed in Table 2.

### AISA extensions

We introduced new accelerator ISA (AISA) extensions in the host core to reduce unnecessary device driver overheads when interfacing accelerators to the servers. These extensions also provide a standard interface for programs to access the accelerator hardware from the user space. This lets programs access the accelerator similar to accessing a processor functional unit or coprocessor. We define two new ISA extensions to create split-instruction semantics (which avoids blocking the processor during the accelerator transaction).

- PUTTXN. This instruction provides a process with an atomic method to send data to an accelerator, along with the context information (returning a unique transaction ID that can be used to query the status asynchronously).
- GETTXN. This instruction provides a process with a method for querying

the hardware for a given transaction's completion status.

We also define an accelerator ID instruction (AID) to enumerate and configure all the available accelerators in the processor.

### AMMU translation services

We designed a simplified memory model to avoid data-copying requirements and enable the accelerators to operate in the virtual-memory domain. The key component of this new model is the accelerator memory management unit (AMMU). The AMMU is integrated on the interconnect and offers address translation services to the accelerators so that they can execute in the virtual-memory domain. The AMMU also lets programs access the accelerator functions directly from the user space and communicate using virtual-memory addresses. When an accelerator tries to access the application memory with a virtual address, the AMMU intercepts the request and automatically translates the virtual address into the corresponding physical address. If the translation isn't present in the operating-system page tables, the AMMU initiates a page fault through an interrupt. The operating system then services the page fault before the AMMU can again attempt the translation.

Table 2 shows the overheads associated with offloading the computationally intensive portions of prominent recognition suites using a traditional device driver approach on the one hand, and using the proposed programming-model-based approach with the AISA and AMMU on the other hand. Table 2 also shows the absolute overhead (in CPU cycles) for the IPD, match, and

GMM accelerators. As is evident from this table, the proposed interfacing model considerably decreases the total data offload cycles when compared with the classical driver model. The reason for this performance benefit is that the overheads for data copying and kernel buffer allocation are avoided when the offload occurs directly from the user space. The only remaining interfacing overheads are those for invoking the offload from software and those due to the hardware delays in queuing up the job for the accelerator. Both of these overheads are insignificant (about 550 cycles) and independent of the nature of the offload as opposed to the driver model.

Other overheads associated with virtual-address-based execution on the accelerators are translation look-aside buffer (TLB) misses and page fault handling. The AMMU keeps a cache of virtual-to-physical address translations in its TLB, which is managed like the processor TLB. Although the analysis of performance data is beyond the scope of this article, our preliminary work in this area suggests that the overheads to service a TLB miss aren't very large when compared to the benefits gained by reducing the software offload path. This is especially true in light of the spatial locality of recognition workloads, which can potentially make intelligent TLB prefetching effective. Increasing the interconnect speeds also reduces the TLB miss service latencies. Page fault servicing should add minimal cost to the performance for recognition accelerator offloading, thanks to its use of source and destination buffers allocated by the offloading program.

The accelerators use *source buffers* to fetch the input data provided by the offloaded program for computation. The nature of recognition workloads dictates that the comparison operations are performed between a target and a source database. Both of these database buffers are prefilled by the software before the control is handed to the hardware accelerators through offloading. The temporal locality makes it improbable for the AMMU to get a page fault when accessing the source buffer.

The accelerators use *destination buffers* to communicate the results of recognition

offloading to software. Some applications allocate a destination buffer on the basis of demand. This implies that, although the pointer to the buffers is valid, the memory doesn't get allocated until the accelerator accesses it to store the result. This can lead to a page fault. However, the number of page faults in such cases is limited (usually just one) because most of the recognition accelerators communicate a "match or no match" result along with an optional index into the database. This result can easily fit into the smallest page allocated by the operating system. The examples discussed in this article, MAR and speech recognition, fall into this category.

As this article shows, the CogniServe recognition server architecture is well-suited for recognition applications and will foster further technology development in this area. With recognition-based applications on smart devices constantly emerging, we will continue studying additional algorithms, and further refine the CogniServe recognition server architecture with the right set of accelerators and associated architectural support.

MICRO

## References

1. L.A. Barroso, J. Dean, and U. Holzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro*, vol. 23, no. 2, 2003, pp. 22-28.
2. G. Takacs et al., "Outdoors Augmented Reality on Mobile Phone Using Loxel-Based Visual Feature Organization," *Proc. ACM 1st Int'l Conf. Multimedia Information Retrieval (MIR 08)*, ACM Press, 2008, pp. 427-434.
3. H. Bay et al., "Speeded-Up Robust Features (SURF)," *J. Computer Vision and Image Understanding*, vol. 110, no. 3, 2008, pp. 346-359.
4. D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int'l J. Computer Vision*, vol. 60, no. 2, 2004, pp. 91-110.
5. S. Srinivasan et al., "Performance Characterization and Acceleration of Optical Character Recognition on Handheld Platforms," *Proc. IEEE Int'l Symp. Workload Characterization (IISWC 10)*, IEEE Press, 2010, doi:10.1109/IISWC.2010.5648852.

6. D.G. Andersen et al., "FAWN: A Fast Array of Wimpy Nodes," *Proc. ACM SIGOPS 22nd Symp. Operating Systems Principles (SOSP 09)*, ACM Press, 2009, pp. 1-14.
7. V. Reddi et al., "Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency," *Proc. 37th Ann. Int'l Symp. Computer Architecture (ISCA 10)*, ACM Press, 2010, pp. 314-325.
8. SeaMicro, <http://www.seamicro.com>.

**Ravi Iyer** is a senior principal engineer and director of the SoC Platform Architecture Research Group at Intel Labs. His research focuses on future system-on-chip (SoC) and chip multiprocessor (CMP) architectures, especially small cores, accelerators, cache and memory hierarchies, fabrics, quality of service, emerging applications, and performance evaluation. Iyer has a PhD in computer science from Texas A&M University.

**Sadagopan Srinivasan** is an engineer at Intel Labs, where he works on architecture and applications for mobile and heterogeneous systems. His research interests include processor architecture, memory subsystems, and performance modeling and analysis. Srinivasan has a PhD in computer engineering from the University of Maryland, College Park.

**Omesh Tickoo** is a research scientist at Intel Labs. His research interests include computer system architecture, multimedia, and networking. Tickoo has a PhD in electrical and computer systems engineering from Rensselaer Polytechnic Institute.

**Zhen Fang** is an engineer at Intel Labs, where he works on architecture and applications for mobile and embedded systems. His research interests include processor architecture, memory subsystems, parallel processing, and performance modeling and analysis. Fang has a PhD in computer science from the University of Utah.

**Ramesh Illikkal** is a principal engineer in the Integrated Platform Research Lab at Intel Labs. His research interests include SoCs,

CMPs, server architectures, virtualization, and memory hierarchies. Illikkal has an MTEch in electronics from Cochin University of Science and Technology in India.


**Steven Zhang** is a senior engineer in the Integrated Platform Research Lab at Intel Labs. His research interests include SoC architecture, reconfigurable computing, and quick prototyping. Zhang has a PhD in communication engineering from Beijing University of Posts and Telecommunications.

**Vineet Chadha** is a research scientist at Intel Labs. His research interests include computer architecture, virtualization, and distributed computing. Chadha has a PhD in computer and information science and engineering from the University of Florida. He's a member of the IEEE Computer Society and the ACM.

**Paul M. Stillwell Jr.** is a senior engineer in the Integrated Platform Research Lab at Intel Labs. His research interests include SoCs and deeply embedded architectures. Stillwell has a BS in electrical engineering from North Carolina State University.

**Seung Eun Lee** is an assistant professor in the Department of Electronic and Information Engineering at Seoul National University of Science and Technology. His research interests include computer architecture, multiprocessor SoCs, networks on chips (NoCs), and very large-scale integration (VLSI) design. Lee has a PhD in electrical and computer engineering from the University of California, Irvine. He's a member of IEEE.

Direct questions and comments about this article to Ravi Iyer, JF2-58, 2111 NE 25th Ave., Hillsboro, OR 97124; [ravishankar.iyer@intel.com](mailto:ravishankar.iyer@intel.com).

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.