High performance computing on SpiNNaker neuromorphic platform: A case study for energy efficient image processing

DOI:
10.1109/PCCC.2016.7820645

High Performance Computing on SpiNNaker Neuromorphic Platform: a Case Study for Energy Efficient Image Processing

Indar Sugiarto, Gengting Liu, Simon Davidson, Luis A. Plana and Steve B. Furber
School of Computer Science, University of Manchester
Oxford Road, Manchester, United Kingdom, M13 9PL
Email: [indar.sugiarto, gengting.liu, simon.davidson, luis.plana, steve.furber]@manchester.ac.uk

Abstract—**This paper presents an efficient strategy to implement parallel and distributed computing for image processing on a neuromorphic platform. We use SpiNNaker, a many-core neuromorphic platform inspired by neural connectivity in the brain, to achieve fast response and low power consumption. Our proposed method is based on fault-tolerant fine-grained parallelism that uses SpiNNaker resources optimally for process pipelining and decoupling. We demonstrate that our method can achieve a performance of up to 49.7 MP/J for Sobel edge detector, and can process 1600 x 1200 pixel images at 697 fps. Using simulated Canny edge detector, our method can achieve a performance of up to 21.4 MP/J. Moreover, the framework can be extended further by using larger SpiNNaker machines. This will be very useful for applications such as energy-aware and time-critical-mission robotics as well as very high resolution computer vision systems.**

1. Introduction

In the last decade there has been a growing interest in bringing green technology into the high-performance computing (HPC) domain. The Top500 Project report shows that supercomputer performance is approximately doubling every year, whilst power consumption is also rising [1]. The energy efficiency of those supercomputers has increased, but at a slower rate than performance. On the other hand, the emergence of neuromorphic technology, *i.e.*, computing platforms inspired by the brain, offers a new paradigm of computation. This technology differs from conventional computer technology not only in its architectural description, but also in that it offers the interesting feature of lower power consumption.

SpiNNaker (Spiking Neural Network Architecture) is a neuromorphic computing system that was built with the motivation to appreciate the marvelous work of the brain. None of the human-made computing technologies can beat the performance of a human brain in terms of communication+computing power. Even Sunway Taihulight, currently the fastest supercomputer in the world [2], has an inferior performance, when measured using the TEPS (Traversed Edges Per Second) metric [3]. SpiNNaker, in contrast, aims to provide relatively less computational power but highly-efficient interconnectivity, similar to the brain itself.

Although initially intended for neuromorphic applications, SpiNNaker has also attracted attention from fields such as robotics, due to its low power consumption. The SpiNNaker chip is designed around an ARM968 processor, the primary market for which is low-power embedded microcontroller applications. A SpiNNaker chip is able to deliver 3600 MOPS (million operations per second) at only 1 Watt in 130nm UMC technology. However, the SpiNNaker chip was designed to be optimal for spiking neural network simulation, with little regard for applications outside of this limited space. Consequently, common functionality that one might expect to find on an general purpose CPU, such as floating point hardware and memory management, are not present. Despite this lack of focus outside of the neural space, the unusual communications fabric and power efficiency of SpiNNaker suggest that it could be an interesting platform on which to evaluate other classes of algorithm. For example, we foresee potential applications of SpiNNaker in computer vision and robotics.

To begin our exploration in this field, in this paper we propose an energy-efficient, high-performance approach to image processing, and evaluate how SpiNNaker performs in this domain. As exemplars that address different aspects of image processing, we demonstrate three algorithms: Sobel edge detector, image smoothing using Gaussian filtering, and image sharpening using histogram equalization.

Our contributions can be summarized as follows:

1) We propose an efficient implementation of fundamental image processing algorithms on a neuromorphic computing platform.
2) We evaluate the efficiency of a scalable, parallel and distributed algorithm running on SpiNNaker.
3) We provide a new benchmark for performance evaluation of many-core neuromorphic platforms.

The rest of this paper is structured as follows: In Section 2 the SpiNNaker architecture and communication network are introduced. In Section 3, we describe our novel method to achieve scalable and efficient image processing using SpiNNaker. Section 4 presents our evaluation of the proposed method. The paper closes with the Conclusions section.

978-1-5090-5252-3/16/$31.00 ©2016 IEEE

2. SpiNNaker Neuromorphic Platform

SpiNNaker is a distributed computing system designed originally to simulate millions of neurons in a spiking neural network model. As a neurally-inspired computing system, it offers opportunities to explore new principles of massively parallel computation which cannot easily be performed by traditional supercomputers.

2.1. SpiNNaker Chips and Machines

The main element of SpiNNaker machines is the SpiNNaker chip. The SpiNNaker chip is a multicore system-on-chip that comprises 18 ARM-968 processor cores. Inside the chip, those cores are surrounded by a light-weight, packet-switched asynchronous communications infrastructure that makes the SpiNNaker machine as a globally asynchronous locally synchronous (GALS) system. Each chip also has a 128 MByte SDRAM (Synchronous Dynamic Random Access Memory), which is physically mounted on top of the SpiNNaker die.

The ARM cores on the SpiNNaker chip do not have floating point unit (FPU), but they can use emulated floating point operation provided by compilers such as GCC (GNU C-compiler). Running such an emulated operation on a core using GCC compiler will yield a performance of up to 9.4 MFLOPS (Million Floating-point Operations Per Second) at a clock frequency of 200MHz. Higher performance can be achieved, theoretically up to 100 MFLOPS, by taking advantage of the special features of the ARM architecture. The work by Iordache [4] gives a good example of how to achieve high performance emulated floating point operation on an integer processor without FPU.

The chip does not employ cache coherence. Instead, each core incorporates two tightly-coupled static memories (SRAM) in a Harvard architecture. As is common in ARM processors, SpiNNaker cores can run in ARM or THUMB mode. Programs written in ARM mode are slightly faster than their THUMB counterpart, however, programs written in THUMB mode have higher code density. Table 1 shows the SpiNNaker ARM core performance benchmarked using Dhrystone, measured in DMIPS (Dhrystone million instruction per second).

TABLE 1. THE SPINNAKER'S DHRYSTONE BENCHMARK AT 200MHZ.

Mode	Unoptimized	Optimized
THUMB	52.7 DMIPS	121.1 DMIPS
ARM	57.5 DMIPS	138.8 DMIPS

As a general purpose computing engine, SpiNNaker is a machine with a large number of homogeneous processing elements. In this paper, we use the term node to refer to a single SpiNNaker chip. The actual number of working cores might be different from chip to chip due to defects during chips fabrication process. During the boot process, any malfunctioning core is excluded from the list of available cores. This is a part of fault tolerance mechanisms in SpiNNaker that take place in several levels to ensure reliability against system failure [5]. In our work, we also applied an additional mechanism such that malfunction cores can also be detected during run time by monitoring their activities. This monitoring task is assigned to the leading core in each node, and the faulty cores will be excluded during the workload distribution to ensure the integrity of the image processing algorithm.

The SpiNNaker architecture is scalable and SpiNNaker machines are classified by the number of processor cores. Table 2 shows the nomenclature used for SpiNNaker machines, where the "10x" machine has approximately 10^x processor cores. Currently, the largest machine in operation consists of 5 105-machine and contains 518400 cores. Fig. 1 shows the SpiNNaker 103 machine used in this paper.

TABLE 2. SUMMARY OF SPINNAKER MACHINE NAMING CONVENTION.

Name	Features
103 machine$^{*)}$	A 48-node board (864 ARM cores) with two 100Mbps Ethernet ports, six 3.1Gbps serial transceivers, and one SpiNN-link port. It requires 12V 6A supply.
104 machine	A single frame incorporating 24 pieces of 103-machine. It has 10,368 ARM processor cores and consumes approx. 1kW of power.
105 machine	A 19" rack cabinet incorporating 5 frames of 104-machine. It has 103,680 ARM processor cores, and requires a 10kW (approx.) power supply.
106 machine$^{+)}$	It comprises 10 cabinets (each a 105-machine). It will have 1,036,800 ARM processor cores, and will require a 100kW (approx.) power supply.

$^{*)}$used in this paper $^{+)}$under construction

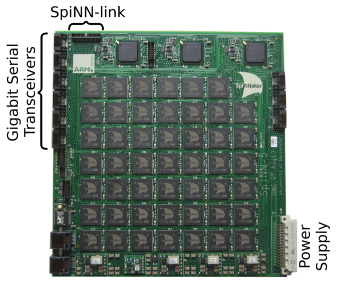


Figure 1. SpiNNaker 103 machine, also known as a SpiNN-5 board.

2.2. SpiNNaker Communication Network

SpiNNaker machines are networks of SpiNNaker chips. A SpiNNaker chip has six bidirectional, inter-chip links that allow the creation of networks with efficient topologies, such as the preferred 2D-torus interconnect. The key component of the SpiNNaker communication infrastructure is its packet-switched network that can distribute short packets in an energy-efficient manner. Packet routing is managed by the asynchronous Network-on-Chip (NoC) which extends seamlessly to the interchip links [6].

Each chip contains a bespoke multicast router with a configurable 3-state, CAM-based look-up table which can send a copy of a packet to any subset of the 18 on-chip cores and the 6 external links concurrently. Thus, an efficient massively distributed computing system can be developed by properly configuring the routing table.

The current addressing scheme in SpiNNaker allows up to 64K chips to be connected in a network, and currently the SpiNNaker machine is targeted to hold up to 57K chips that will comprise 1,036,800 processor cores and 7TeraBytes of RAM. With this setup, it is arguable that SpiNNaker machine has a promising feature for a high performance computing machine. Currently, a half million-core SpiNNaker machine has been built and goes under intensive hardware testing in the University of Manchester.

There are four types of data packet that can be circulated in a SpiNNaker machine using the chip-to-chip interconnect: nearest-neighbor (NN) packets routed to any subset of the six neighbor chips, point-to-point (P2P) packets routed by destination addresses, multicast (MC) neural packets routed by source addresses, and fixed-route (FR) packets routed by the contents of a register set on a chip by chip basis. Of these, only the MC and P2P packet types are of interest to applications, the former being the principal mechanism for cores to exchange small pieces of data and the latter acting as the layer upon which long messages (including traffic to and from the host) are sent using a special protocol called SDP (SpiNNaker Datagram Protocol), to be described next.

Table 3 summarizes the use of those packets for the proposed strategy in our work.

TABLE 3. SUMMARY OF SPINNAKER PACKET TYPES AND THEIR USAGE IN OUR PROPOSED METHOD.

Type	Features	Usage
P2P/SDP	direct point-to-point, convey large payload, slow	receive/send data/frames from/to host-PC
MC/FR	multiple destination, convey small payload, very fast	coordination among cores, workload distribution

In supercomputers, the network is generally designed to handle the transfer of large bursts of data. The communication between nodes is managed using standard protocol such as MPI (Message-Passing Interface). In contrast, SpiNNaker is optimized for the energy-efficient transmission of short messages. The current SpiNNaker kernel, however, does not support MPI. The SpiNNaker Datagram Protocol (SDP) was designed for the inter-chip transmission of data blocks. If an MPI-style communication is required, SDP can be used to mimic such a protocol.

An SDP packet can be embedded in a UDP/IP packet, with appropriate SDP header information, as shown in Fig. 2. In our work, the main use of SDP is for transferring image data between the host-PC and the SpiNNaker machine.

3. Strategies for Scalability and Efficiency for Image Processing

High-performance computer vision using parallel processing on machines with general purpose microprocessors has been investigated since 1950 [7]. In general, high-performance image processes exploit the fact that many image processing operations are inherently parallelizable. The parallel computations are performed using local neighborhood operations, which are almost always independent. However, their implementation is often criticized due to the inappropriate organization of the algorithms that leads to significant communication overhead between the parallel jobs [8]. In recent years, the advancement of graphic card technology has contributed even more to the growth of high performance image processing. Parallel processing using GPGPU (General Purpose computing on Graphics Processing Unit) often produces impressive results that cannot be obtained using conventional multi-core computers [9], [10]. Although the implementation of parallel image processing on multi-core computers and on modern GPUs bears little resemblance to each other, they still share the same burden: they consume a considerable amount of power.

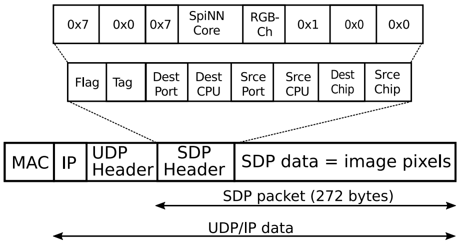


Figure 2. SDP packet format and its usage for transferring image pixels from host-PC to the SpiNNaker.

As we have described in Section 2, the strength of the SpiNNaker machine lies not only in its low-power cores, but also in its bespoke energy-efficient networking. The successful implementation of any parallel-and-distributed system depends on the strategy for distributing the computation load. Hence, the challenge of using SpiNNaker for high-performance computing lies mainly on how to optimize the use of its communication network.

In our image processing work, we use MC packets to distribute pixel values as well as for coordination among neighborhood operations. An MC packet contains an 8-bit header and a 32-bit key information. It can be extended with a 32-bit payload. Fig. 3 shows the structure of this MC protocol and how we manipulate it to carry messages or useful commands in our algorithms.

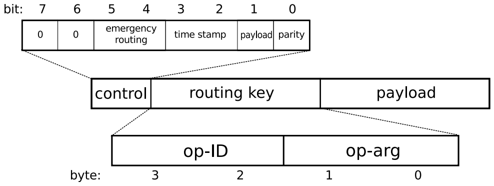


Figure 3. MC neural packet format and its usage for image processing.

The experiments reported in this paper were executed in a SpiNNaker 103 machine. Fig. 4 shows the configuration of the experimental setup. Sending and receiving images to/from the SpiNNaker machine are performed using SDP messages, whilst distributing images and processing them inside the SpiNNaker machine use MC packets. SpiNNaker 103 machines use the 100 Mbps Ethernet link for communication with the host-PC, and we send and receive images through this link. We can also use a bespoke asynchronous SpiNNaker link, with much higher throughput than the Ethernet link, to send the resulting image directly to a monitor via an external FPGA board.

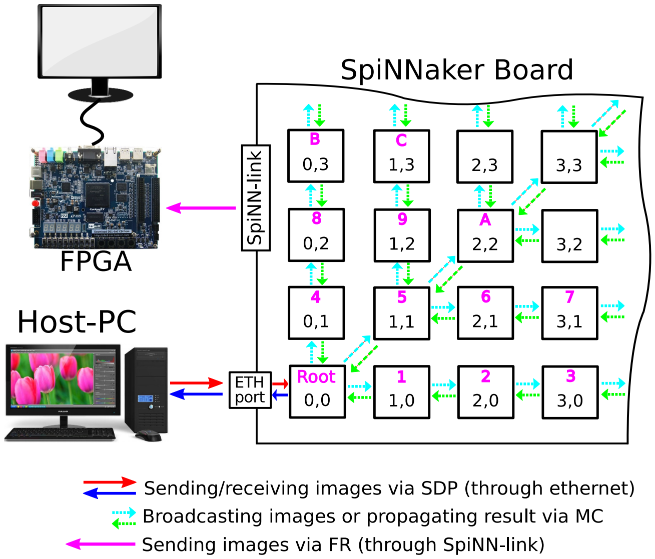

SpiNNaker Board
FPGA
Host-PC
SpiNN-link
ETH port
B 0,3
C 1,3
2,3
3,3
8 0,2
9 1,2
A 2,2
3,2
4 0,1
5 1,1
6 2,1
7 3,1
Root 0,0
1 1,0
2 2,0
3 3,0
Sending/receiving images via SDP (through ethernet)
Broadcasting images or propagating result via MC
Sending images via FR (through SpiNN-link)

Figure 4. The experiment setup for image processing on SpiNNaker. The host-PC sends image's pixels to SpiNNaker via SDP. The root-node receives the SDP packets, converts them into MC packets, and broadcasts them to all nodes. When the processing is done, each node will send the resulting part via SDP directly to either the host-PC or the external FPGA board.

The overall process of the image processing implementation in our experiment is shown in Algorithm 1.

3.1. Parallel Algorithm for Image Processing

In general, image processing can be performed in different levels: from the lowest level that works with image pixels to the highest level that exploits hidden features in an image to gain knowledge such as object recognition and semantic description. In this paper, we demonstrate two important aspects of low-level parallel processing. We use a task graph representation to describe the mechanism and the behavior of our method. In this task graph formalism, we regard a node as a vertex in the graph that is responsible for carrying out an atomic task. Hence, we can represent our whole algorithm as a DAG (Directed Acyclic Graph).

Based on locality, parallel image processing can be loosely classified into two categories: processing with locally independent neighborhood operations, and processing with global dependency of neighborhood operations. In the first category, the neighborhood operations, which usually employ a processing window of size 3 x 3 or 5 x 5, can be performed completely in parallel and asynchronously without one operation affecting the others. We give two examples of this category: image smoothing using Gaussian filtering, and edge detection using Sobel and Laplace operators. In the second category, the operations might need synchronization due to global linkage to a certain attribute. We present image sharpening using histogram equalization as an example of this category.

Algorithm 1 Overall image processing experiment

```
Each node counts how many cores are available
Host-PC sends image size to SpiNNaker
Each core in SpiNNaker computes its working load
Distribute image in SpiNNaker:
for all available cores in the root-node do
    for all color channels do
        fetch pixels data from SDP packets
        compute gray scaling
        broadcast grayscale pixels to all nodes
    end for
end for
Perform image processing:
for all working cores do
    fetch pixels from SDRAM via DMA
    convolute pixels
    put the result to SDRAM via DMA
end for
Send result:
for all nodes do
    for all lines do
        fetch lines from SDRAM via DMA
        put into SDP packets and send them out
    end for
end for
return  SpiNNaker processing time
```

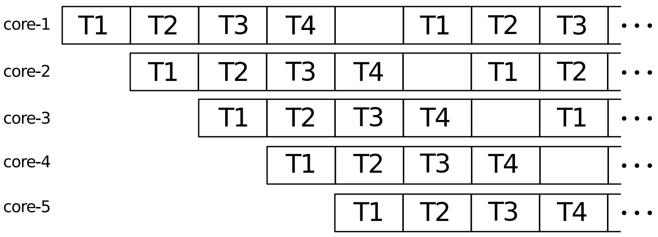
In addition to these examples, we also consider the parallelizing strategy for image retrieval and pre-processing, specifically for grayscale processing. Although our methods also work for color (RGB) images, we emphasize grayscale processing because it is adequate for many high-level image processing and computer vision task, such as in the object recognition.

3.1.1. Sending and pre-processing images. The sending and pre-processing of images can be performed in a parallel fashion as follows: images are sent from the host-PC to the root node, (*i.e.*, the node with ID-0 at coordinate 0, 0), and are pre-processed there using the following four tasks:

1) **T1** - retrieving RGB pixel data by extracting them from SDP packets sent by host-PC.
2) **T2** - calculating the grayscale values for the retrieved RGB pixels.
3) **T3** - distributing the grayscale pixels as MC packets to other nodes in the system.
4) **T4** - computing the number of occurance for the grayscale pixels that will be used later for histogram equalization.

To optimize the performance, these tasks are arranged as a pipeline, and use several available cores in the root node.

Fig. 5 shows the pipeline mechanism using five cores. Using this mechanism, sending images over the Ethernet link can achieve a throughput close to 80 Mbps (the maximum of 100 Mbps cannot be achieved due to SDP-related overheads).



core-1	T1	T2	T3	T4		T1	T2	T3	···
core-2		T1	T2	T3	T4		T1	T2	···
core-3			T1	T2	T3	T4		T1	···
core-4				T1	T2	T3	T4		···
core-5					T1	T2	T3	T4	···

Figure 5. Pre-processing pipeline for sending image, grayscaling, distribution, and computing pixels histogram.

A similar pipeline mechanism is implemented in other nodes on the SpiNNaker board, but without tasks T1 and T3. Also, task T2 is modified so that the cores on those nodes will not compute the grayscale values again, but only store the retrieved pixels broadcast by the root node.

3.1.2. Image Smoothing and Edge Detection. Image smoothing and edge detection can be performed using convolution. For edge detection, we use the Sobel operator with two 3 x 3 kernels that are convolved with the input image. The two kernels represent derivative approximations: G$_x$ for horizontal changes, and G$_y$ for vertical ones. These two kernels are as follows [11]:

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \mathbf{G_y} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (1)$$

Image filtering is commonly used in edge detection preprocessing to smooth the image by filtering out noise that are usually related to high-frequency components of the image. We used a Gaussian filter with a 5 x 5 kernel size and $\sigma = 1.0$. The discretized form of this Gaussian kernel is as follows [11]:

$$\mathbf{G} = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \quad (2)$$

In our implementation, the processes are executed in parallel and independently from each node. After pixels broadcasting is completed, the leading core in the root node will broadcast an MC packet to all nodes. For this purpose, we configure the routing table in each node such that the node will propagate the message to its adjacent node, and we have to make sure there is no duplication of this message. The resulting routing flow, for an example network with 13 nodes, is shown in Fig. 6. Each node is labelled according to its position in the board, as shown in Fig. 4. Once a node has completed its operation, it sends an MC packet back to the root node using the same route.

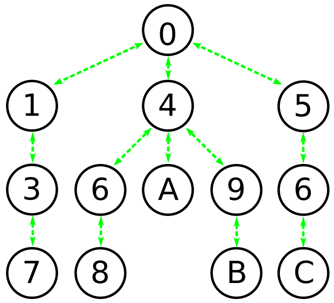


Figure 6. Messages propagation for image filtering and edge detection. The convolution operation in each node is independent from other nodes. The route is optimized such that the message travels on a shortest path.

3.1.3. Image Sharpening. Unlike the smoothing and edge detection processes, image sharpening using histogram equalization cannot be performed directly using convolution. Each node computes only the histogram that corresponds to its working load, and the network needs to know the global knowledge about these partitioned histogram. To collect the histogram partitions from all nodes, we employ SDP communications in a binary tree structure shown in Fig. 7. Intermediate nodes aggregate partions on their way to the root node. For example, nodes 7 and 8 send their histogram partition to node 3, which merges those incoming histogram partitions with its own, and then propagates the combined result to node 1. This process continues until nodes 1 and 2 propagate their histogram partitions to the root node. The root node merges the incoming histogram with its own, and then broadcasts back the unified histogram to all nodes using MC packets. Once a node receives a complete histogram it starts normalizing its own workload. Each node then sends an MC packet back to the root node using the route in Fig. 6 to report the completion of the histogram equalization process.

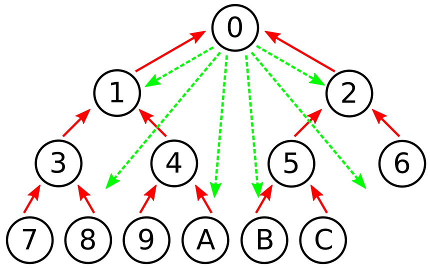


Figure 7. Messages propagation for histogram equalization. In this process, we have to use two communication protocols: SDP (red) for propagating histogram partitions, and MC (green) for broadcasting final histogram data.

4. Evaluation and Discussion

For the experiment, we sent images of various sizes ranging from VGA to UXGA resolution and measured SpiNNaker performance and power consumption for each image processing task. Regarding power consumption, SpiNNaker chips have frequency scaling capability, with cores operating reliably up to 255 MHz. Spiking Neural Network (SNN) simulations are usually run at a core clock frequency of 200MHz, which is adequate to meet their real-time deadline and keeps power consumption low. In our experiments we measure power consumption at two different frequencies, 200MHz and 250MHz, to evaluate the different performance/power consumption scenarios.

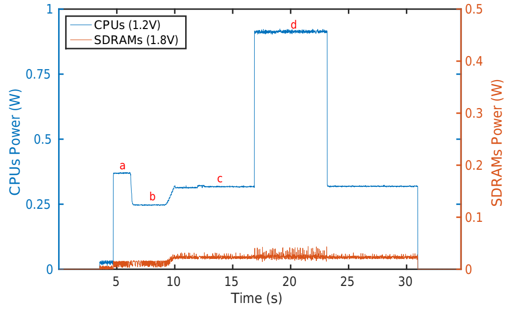


Figure 8. Power consumption of a SpiNNaker chip. During boot, (a), the SpiNNaker chip consumes 370mW. Without any application running, (b), the SpiNNaker chip consumes 250mW. When our image processing program is loaded, (c), it consumes only 320mW. When the program starts intense computation, (d), the power rises to 930mW. During this execution, the power consumed by the SDRAM is only 25mW.

In normal (SNN) operation at 200MHz, after system boot and without any program running, a SpiNNaker chip consumes only 250mW. Increasing the clock frequency to 250MHz, will raise the power consumption to about 300mW. When the application is loaded the processor cores start to draw more power. During an intense computation, if all cores are used, the total power consumed by the chip can rise up to 950mW. Fig. 8 shows the power consumption profile of a SpiNNaker chip when we run the experiment with edge detection processing on a single node.

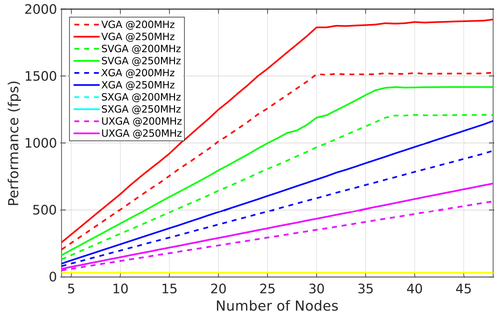


Figure 9. Edge detection performance.

We evaluate the computation speed of our programs with respect to the number of nodes and the core clock frequency. Fig. 9 shows the performance of SpiNNaker 103 when running the edge detection program. It shows the performance for different clock frequencies and image resolutions. It can be seen that performance scales linearly with the number of nodes used. However, for VGA and SVGA resolutions, performance stops increasing at 30 and 37 nodes, respectively. This is the result of a constraint in our algorithm: each line of the image will be processed by one core as a maximum. Hence, an image with VGA resolution of 640 x 480 pixel, can only utilize 480 cores, which can be provided by 30 nodes. Thus, nodes 31 to 48 will not be used, and the performance will be steady at this position.

Overall, we can see that SpiNNaker performance is far above the real-time processing requirement, which is 30 frames per second (fps) shown as the yellow line at the bottom of the figure. Increasing the clock frequency is also the main contribution to increasing performance. However, as we explain below, using higher frequency consumes more power.

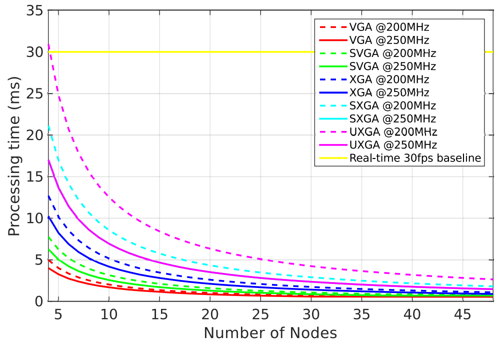


Figure 10. Gaussian filter performance.

Fig. 10 shows the performance of SpiNNaker 103 when running the Gaussian image filtering program. In this figure, we plot SpiNNaker performance as processing time in milliseconds. Similar to the edge detection result, performance also scales up with the number of nodes. If this result is plotted as fps, then we will get a plot similar to Fig. 9. Indeed, the processing power for these two convolutional processes is very similar, as we can see in Fig. 12.

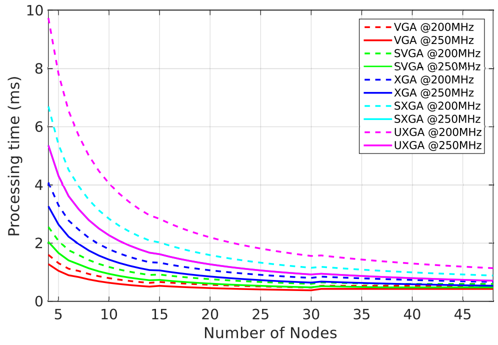


Figure 11. Histogram equalization performance.

Fig. 11 shows the performance of SpiNNaker 103 when running the histogram equalization program. The histogram equalization requires less computation than the convolution operation, hence, it can be performed faster than either the Gaussian image filtering or the edge detection program. Most of the processing time in Fig. 11 was contributed by the propagation delay of the SDP messages that carry histogram partitions from each node. Once the full histogram is received, the normalization computation using that histogram runs very fast; it takes only 191 microseconds for a 640 x 480 image and 798 microseconds for a 1600 x 1200 image.

Combining computation speed and power consumption, we obtain the overall processing power of our image processing programs on SpiNNaker 103. Based on the measured speeds as presented before, we expect to see the trend of increasing performance as the number of nodes used for image processing increases. Fig. 12 shows a summary of this trend.

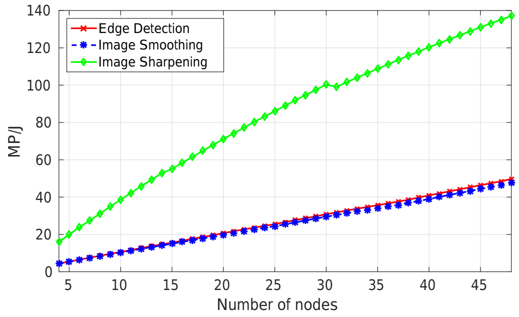


Figure 12. Computing power performance.

The performances of both image smoothing and edge detection programs scale up linearly almost at the same pace. But the performance of the image sharpening program is far above those two. The reason is because image sharpening, which employs histogram equalization, consumes far less power than the other two programs, which require image convolution. We also observe discontinuities at several places, most significantly at 15 and 30 nodes. This happens because the number of SDP message propagation (see Fig. 7) changes at that point. As we described earlier, SDP propagation takes a considerable amount of time, whereas the equalization computation takes much shorter time. Nevertheless, the image smoothing performance seems to scale up almost linearly.

TABLE 4. Performance on VGA image using 5 x 5 filter kernel

Platform	**frame per seconds (fps)**
FPGA	344
CPU	400
GPU	3000
SpiNNaker-103	$1923^{*)}$ - $2976^{+)}$

$^{*)}$current result $^{+)}$estimation

For a comparison of processing speed with that of other implementations we refer to the work by Asano *et al.* [12]. In their work, they used three different platforms: CPU, GPU, and FPGA. For the CPU platform, they used an Intel Core 2 Extreme QX6850 (3GHz, quad-cores). For the GPU, they used a XFX GeForce 280 GTX 1024MB DDR3 (1.3GHz, 1.1GHz). And for the FPGA, they used a Xilinx XC4VLX160 running at 100MHz. They reported that the GPU implementation is superior to the other implementations in all kernel sizes (but decreases exponentially as the size of the kernel increases), whereas the CPU and the FPGA performance are comparable when the kernel size is 5 x 5. For a fair comparison with our work, we also use a 5 x 5 kernel and the results are shown in Table 4.

From the table, we can see that SpiNNaker 103 runs faster than CPU and FPGA, but GPU is the fastest platform. The current measured speed of SpiNNaker 103 is 1923 fps, but we expect to achieve a speed of up to 2976 fps if we modify our program to alleviate the current constraint that each image line will be processed only by one core. If we use, for example, two cores to process a single line, then we will obtain faster response; eventually we can get 2976 fps. However, we currently do not use this method but consider it as a future extension.

Regarding the superiority of the GPU performance shown in Table 4, one reason that can be inferred from the work [12] is that the filter's kernel was applied to each pixel in the image independently without using shared variables. Given that the GPU uses a very high frequency for both its processing elements and its DDR3 memory, it is clear that it will outperform the other platforms. As a comparison, each SpiNNaker core runs at 250MHz and its SDRAM runs at only 133MHz. In contrast, the GPU GeForce 280 GTX used in [12] runs at 1.3GHz for its cores and at 1.1GHz for its DDR3 memory. Furthermore, the power consumption is not considered in their work, and we believe that the GPU power consumption is also high.

TABLE 5. Power consumption comparison

Image Resolution	**CPU J**	**GPU J**	**FPGA mJ**	**SpiNNaker 103 mJ**
512 x 512	4.2	0.5	1.6	21.9
1024 x 1024	14.8	1.5	6.4	50.7
1476 x 1680	39.8	3.4	15.0	121.3
3936 x 3936	229.0	15.0	93.6	723.5

To compare SpiNNaker 103 power consumption with that of other platforms we refer to the work by Possa *et al.* [11]. They also use three different platforms: CPU, GPU, and FPGA. Unfortunately, they implemented the Canny edge detection algorithm. Since we did not implement the same edge detection algorithm, we use estimated values for the comparison. To estimate our results for the Canny edge detection we can combine our Gaussian filtering and Sobel edge detection, given that both techniques are used by the Canny edge detection. Hysteresis thresholding is an additional component of the Canny edge detection algorithm. It is a simple operation and, given that we did not implement it, we replaced it with the normalization operation used in our histogram equalization program to complete the simulated Canny edge detection. The result is presented in Table 5. We can see that the power consumption of SpiNNaker is higher than the FPGA, but it is much lower than the CPU and the GPU. However, the difference on power consumption between the FPGA and the SpiNNaker might be negligible if we also take into account the power consumed by the supporting peripherals in the FPGA board.

As a summary, we combine the measurement from the processing speed and the power consumption of our simulated Canny edge detector on SpiNNaker for several image resolutions. The result is presented in Table 6. It seems that the performance will be steady at about 21 MP/J for higher resolution images.

TABLE 6. SUMMARY OF SPINNAKER PERFORMANCE.

Image Size	**512x512**	**1024x1024**	**1476x1680**	**3936x3936**
MP/J	12.0	20.7	20.5	21.4

5. Conclusions

This paper presented a strategy to use the SpiNNaker neuromorphic computing platform for applications beyond the spiking neural network simulations for which it was originally designed and built. In particular, we developed a method to implement massively parallel and distributed processing for energy-efficient, high-performance image processing. This novel method relies on the full utilization of the SpiNNaker networking resources that allows our program to optimally distribute the computation loads.

As indicated earlier, we used a SpiNNaker 103 machine, with 48 nodes and 6 GB of SDRAM, to implement a series of image processing experiments. We ran the SpiNNaker machine at a moderate working frequency of 200 MHz, but during processing the frequency may be altered up to 250 MHz adaptively. This scenerio will keep power consumption between 0.25 and 0.9 Watt per chip. During the experiments, we sent images of various sizes, from VGA to UXGA resolution, and recorded the SpiNNaker performance executing several image processing algorithms.

We presented three examples of image processing applications: image smoothing, edge detection, and image sharpening. These three example applications demonstrated the communication+computing power of SpiNNaker. For the image smoothing, using a 5 x 5 Gaussian kernel with $\sigma = 1.0$, SpiNNaker performance was measured at up to 47.6 megapixels per second per Joule (MP/J). For the edge detection, using a Sobel operator, the performance was measured as 49.7 MP/J. For image sharpening using histogram equalization, the performance was measured as 131.7 MP/J. We also implemented a simulated Canny edge detector by combining several elementary processing and the SpiNNaker achieves 21.4 MP/J for a 4K image.

In general, we conclude that the SpiNNaker delivers an impressive performance, and offers an alternative solution as a platform for massive image processing. Conceptually, we can achieve higher performance by using larger SpiNNaker systems, such as a 104 machine or even a 105 machine. We believe that fast response and low power consumption, as demonstrated by the SpiNNaker machine 103 in our experiment, will be very useful for applications such as energy-aware and time-critical-mission robotics. It can also be used for extremely high-resolution computer vision systems such as in astronomical image analysis. With these results, we demonstrate that SpiNNaker systems have promising future application as green neuromorphic-based supercomputers.

Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) through the Graceful project (grant EP/L000563/1). The design and construction of the SpiNNaker machine was also supported by EPSRC (grants EP/D07908X/1 and EP/G015740/1). Ongoing development of the software is supported by the EU ICT Flagship Human Brain Project (FP7-604102), and LAP, SD and SBF receive support from the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement 320689.

References

[1] M. Feldman and TOP500.org, "Energy-efficient supercomputing comes of age with TOP500-Green500 merge," 06 2016.

[2] TOP500.org, "Top500 list - june 2016," 06 2016.

[3] F. Checconi and F. Petrini, "Traversing trillions of edges in real time: Graph exploration on large-scale parallel machines," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pp. 425–434, May 2014.

[4] C. Iordache and P. T. P. Tang, "An overview of floating-point support and math library on the Intel® XScale™ architecture," in *Proc. 16th IEEE Symposium on Computer Arithmetic*, pp. 122–128, June 2003.

[5] J. Navaridas, S. Furber, J. Garside, X. Jin, M. Khan, D. Lester, M. Lujn, J. Miguel-Alonso, E. Painkras, C. Patterson, L. A. Plana, A. Rast, D. Richards, Y. Shi, S. Temple, J. Wu, and S. Yang, "SpiNNaker: Fault tolerance in a power- and area- constrained large-scale neuromimetic architecture," *Parallel Computing*, vol. 39, no. 11, pp. 693–708, 2013.

[6] L. Plana, J. Bainbridge, S. Furber, S. Salisbury, Y. Shi, and J. Wu, "An on-chip and inter-chip communications network for the SpiNNaker massively-parallel neural net simulator," in *The Second ACM/IEEE International Symposium on Networks-on-Chip*, (Newcastle, UK), 2008.

[7] I. Pitas, *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*, ch. 1. John Wiley & Sons Ltd., 1993.

[8] J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, "Decoupling algorithms from schedules for easy optimization of image processing pipelines," *ACM Trans. Graph.*, vol. 31, pp. 32:1–32:12, July 2012.

[9] S. Philip, B. Summa, V. Pascucci, and P. Bremer, "Hybrid CPU-GPU solver for gradient domain processing of massive images," in *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, (Taiwan), pp. 244–231, December 2011.

[10] K. Ogawa, Y. Ito, and K. Nakano, "Efficient canny edge detection using a GPU," in *2010 First International Conference on Networking and Computing*, (Hangzhou, Zhejiang, China), pp. 279–280, October 2010.

[11] P. R. Possa, S. A. Mahmoudi, N. Harb, C. Valderrama, and P. Manneback, "A multi-resolution FPGA-based architecture for real-time edge and corner detection," *IEEE Transactions on Computers*, vol. 63, pp. 2376–2388, October 2014.

[12] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *2009 International Conference on Field Programmable Logic and Applications*, pp. 126–131, August 2009.