MEDICS: Ultra-Portable Processing for Medical Image Reconstruction

Ganesh Dasika, Ankit Sethia, Vincentius Robby, Trevor Mudge, and Scott Mahlke
Advanced Computer Architecture Laboratory
University of Michigan - Ann Arbor, MI
{gdasika,asethia,acolyte,tnm,mahlke}@umich.edu

ABSTRACT

Medical imaging provides physicians with the ability to generate 3D images of the human body in order to detect and diagnose a wide variety of ailments. Making medical imaging portable and more accessible provides a unique set of challenges. In order to increase portability, the power consumed in image acquisition – currently the most power-consuming activity in an imaging device – must be dramatically reduced. This can only be done, however, by using complex image reconstruction algorithms to correct artifacts introduced by low-power acquisition, resulting in image processing becoming the dominant power-consuming task. Current solutions use combinations of digital signal processors, general-purpose processors and, more recently, general-purpose graphics processing units for medical image processing. These solutions fall short for various reasons including high power consumption and an inability to execute the next generation of image reconstruction algorithms. This paper presents the MEDICS architecture – a domain-specific multicore architecture designed specifically for medical imaging applications, but with sufficient generality to make it programmable. The goal is to achieve 100 GFLOPs of performance while consuming orders of magnitude less power than the existing solutions. MEDICS has a throughput of 128 GFLOPs while consuming as little as 1.6W of power on advanced CT reconstruction applications. This represents up to a 20X increase in computation efficiency over current designs.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems

General Terms

Design, Performance

Keywords

Low power, Medical imaging, Operation chaining, Stacked DRAM, Stream processing

1. INTRODUCTION

Medical imaging is one of the most effective tools used in modern medicine to aid physicians in diagnosing and analyzing ailments. Computed tomography, or CT, employs geometry processing to generate a three-dimensional image of the inside of an object from a large series of two-dimensional x-ray images taken around a single axis of rotation. More than 62 million scans are ordered each year to detect ailments ranging from brain tumors and lung disease to guiding the passage of a needle into the body to obtain tissue samples [3]. Compared to traditional 2D x-rays, CT has inherent high-contrast resolution to accurately detect differences in tissue density of less than 1%. Further, the data is highly flexible in that it can be aggregated and viewed in all three planes depending on the diagnostic task. Other popular medical imaging techniques use varying methods to acquire images, e.g., other forms of radiation or radio frequency signals, including single photon emission computed tomography (SPECT), positron emission tomography (PET), and magnetic resonance imaging (MRI).

From a computer architecture perspective, the challenging aspect of medical imaging is the vast amount of computation that it requires. This computation is floating-point-intensive and utilizes a large amount of data. Figure 1 presents the performance requirements and power envelopes of large scale imaging systems as well as portable (bed-side or in-field) systems. Here, the term image reconstruction (IR) is used to encompass the key algorithms found in most variations of medical imaging. From the figure, the performance of advanced imaging used in non-portable systems ranges from 900 GFLOPs to nearly 10 TFLOPs. Portable IR requires an order of magnitude less performance, but also has a substantially lower power budget to operate in a less tethered environment. Figure 1 also presents the peak performance and power of several commodity processors for comparison including processors from Intel, IBM and Nvidia.

Current medical imaging compute substrates: Conventional CT scanners and MRI systems use a combination of general-purpose x86 processors, ASICs, and FPGAs to perform the necessary computation. The JXT6966 from Trenton systems [27] is a board consisting of multiple Core i7-class processors; Texas Instruments has a number of comprehensive solutions which use a combination of analog components to control the x-ray emitters and detectors, and fixed-point DSPs for image reconstruction [24]; and Nvidia GPGPUs have been used to accelerate MRI reconstruction [22]. These solutions all have their drawbacks. The TI solutions do not support floating-point computation. The x86-based solutions require turnaround times of many hours for the advanced IR algorithms that researchers propose [26]. As a result, many developers have turned to general-purpose graphics processing units, or GPGPUs, which are capable of delivering the requisite performance.

GPGPUs, however, have a large disparity between their compute capabilities and memory bandwidths. For instance, the latest generation of Nvidia GPGPUs, the GTX285, has a peak performance of 1,063 GFLOPs but can transfer only up to 159 GB/s of data. This works out to an average 0.15 bytes of data per FP operation. For graphics applications, such a ratio is sustainable, but most IR algorithms require an order of magnitude more memory bandwidth due to the data-intensive nature of the computation. The resultant bandwidth-limited performance is considerably less than the reported peak performance. Power consumption, too, is an issue for many of the existing solutions.

Low-radiation and portable medical imaging: Portable medical imaging is an area of growing interest in the medical community for a variety of reasons. Recent medical studies [29, 5, 17] have shown a marked improvement in patient health when using portable CT scanners and MRI machines, especially in emergency and critical cases. The National Institute for Neurological Disorders and Stroke recommend that patients displaying signs of a severe stroke undergo a CT scan to examine if they would be eligible for thrombolytic therapy – injection of a blood-clot dissolving medication. However, there is a short window of 3 hours for the treatment to be applied. In [29], the effective usage of portable CT scanners allowed patients to be examined more quickly after their arrival at a hospital and resulted in an 86% increase in the number of patients who were eligible for thrombolytic therapy.

In [5], the authors examined the number of medical and technical complications that occurred in medium and high-risk neurosurgery intensive-care unit patients while they were being transporting to CT scanning rooms. Using mobile CT scanners and bringing the scanners to the patients, rather than vice-versa, cut down not only the time required (between 40% and 55%) and the number of hospital personnel required to perform the scan but, most importantly, reduced the number of complications between 83% and 100%.

Conventional CT scanners require a very large amount of power to operate, the majority of which is for the x-ray emitters themselves which consume several kilowatts of power. However, due to the ever increasing number of CT scans performed on patients, there is growing concern about the effects of elevated radiation exposure [3] and, consequently, increased interest in reducing the intensity and power of the x-rays [13]. One approach is using carbon nanotube-based x-ray emitters which use just a few milliwatts of power [21, 32]. Using low-power and low doses of x-rays, however, requires more compute-intensive, iterative techniques to compensate for the associated artifacts [6, 26]. Therefore, reducing the supply voltage and clock frequency of high-performance processors is not an appropriate solution to reduce the power of the reconstruction engine as these techniques reduce performance as well. Essentially, efforts made to reduce the power consumption of x-ray emission and detection are increasing the power requirements of the reconstruction engines, to the point that the computational devices used for image reconstruction have become the dominant power-consuming components.

A new domain-specific design: To overcome the problems of high power consumption and insufficient memory bandwidth, this work presents an architecture and system design that is targeted for portable medical imaging. Our design, named MEDICS (Medical Imaging Compute System), utilizes three critical technologies to achieve its objectives: (1) a 2D datapath comprised of a chained, wide-SIMD floating-point execution unit to efficiently support the commonly occuring computation subgraphs while minimizing accesses to the register-file; (2) image compression units to compress/decompress data as it is brought on-chip to maximize the available off-chip memory bandwidth; and (3) a memory system consisting of a 3D stacked DRAM and input/output streaming buffer to sustain high utilization of a wide-SIMD datapath. Power-efficiency is also garnered by using compile-time software pipelining to hide memory and datapath latencies, thereby eliminating the need for sustaining a large number of contexts found in some high-performance processors like GPGPUs. Overall, MEDICS achieves 128 GFLOPS while consuming as little as 1.6W on state-of-the-art CT reconstruction algorithms.

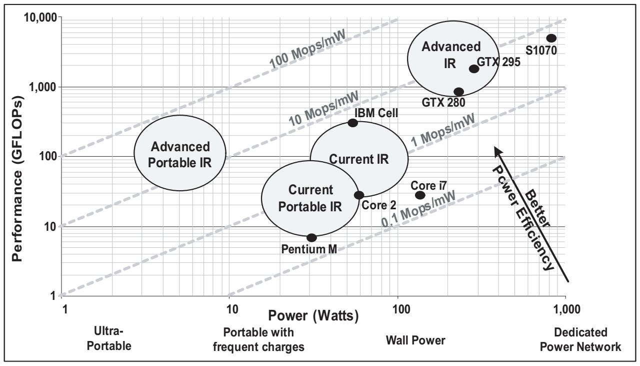


Figure 1: Performance and power requirements for the domains of current and advanced image reconstruction techniques in both tethered and untethered environments. Diagonal lines indicate "Mops/mW" performance/power efficiency. For comparison, the peak performance and power of several commercial processors and GPGPUs are provided: Intel Pentium M, Intel Core 2, Intel i7, IBM Cell, Nvidia GTX 280, Nvidia GTX 295, and Nvidia Tesla S1070.

2. COMPUTATIONAL REQUIREMENTS OF IMAGE RECONSTRUCTION

The fundamental problem in tomographic image reconstruction is illustrated in Figure 2. Figure 2(a) sketches the general concept of how a CT scan occurs. The process begins with x-rays (dotted arrows) being shot from multiple directions at the object. The detector on the opposite side of the x-ray emitter can only measure x-ray attenuation, so it detects very few of the x-rays passing through the opaque cylinder (represented by the dark lines), but detects more of the x-rays passing through the transparent cube (represented by the light line).

Successive scans (or slices) all around the subject at various angles leads to the problem presented in Figure 2(b). Here, each arrow represents a path through the matrix and the number next to it is the sum of all the numbers in the elements that the arrow passes through. Using this information, the values in the matrix are populated. Similarly, in tomography, the matrix is the cross section of interest in the human body, each arrow represents one x-ray and the number next to the arrow represents the attenuation measured by the corresponding cell in the detector array. Using the x-ray positions, angles and attenuations, the reconstruction algorithms have to compute the densities and sizes of all the elements in the matrix in order to provide a two-dimensional image of the area of interest.

The total data per scan that reconstruction algorithms have to process is quite large. For instance, the raw data collected by a recent generation 3D multi-slice helical CT system has the following dimensions:

N_{s}	888	samples per detector row
N_{a}	984	projection views per rotation
N_{t}	64	detector rows
N_{r}	10	rotations (turns of helix)

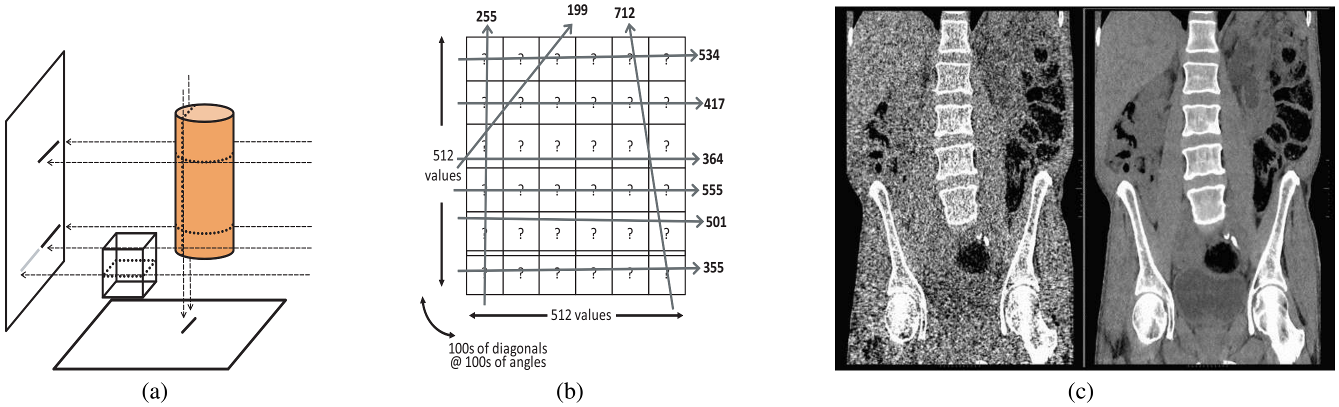


Figure 2: Capturing and deciphering x-rays for tomographic image reconstruction. (a) Detected x-ray attenuations vary based on the density of the object; the opaque cylinder allows much fewer x-rays to pass through it than the transparent cube. (b) Internal density values must be computed using x-ray attenuations measured by the detector array. (c) A slice of 3D helical x-ray CT scan reconstructed by the conventional FBP method (left) and by a MBIR method (right). For these thin-slice images, the MBIR method exhibits much lower noise than the FBP images, enabling better diagnoses.

A raw data set of the above dimensions is called a sinogram, and iterative reconstruction methods need three sinograms: the log transmission data, the statistical weighting associated with each measurement, and a working array where predicted measurements are synthesized based on the current estimate of the object at a given iteration. Each of these three sinograms are stored as single-precision, 4-byte floating-point (FP) values, so the typical minimum memory needed for the data is

$$3 \cdot 4 \cdot N_\mathrm{s} \cdot N_\mathrm{t} \cdot N_\mathrm{a} \cdot N_\mathrm{r} \approx 6.4\ \mathrm{GB}.$$

From that data, one reconstructs a 3D image volume that is a stack of $N_\mathrm{z} \approx 700$ slices, where each slice contains $N_\mathrm{x} \times N_\mathrm{y}$ pixels, where typically $N_\mathrm{x} = N_\mathrm{y} \approx 512$. These image values are also stored as single precision FP numbers, so the memory required for a single 3D image volume is

$$4 \cdot N_\mathrm{x} \cdot N_\mathrm{y} \cdot N_\mathrm{z} \approx 700\ \mathrm{MB}.$$

Advanced iterative algorithms for image reconstruction require 2 to 5 arrays of that size, so several GB of RAM are needed for these 3D image volumes. Further, since the imaging data is iterated over several times, the time taken to access the data must be kept small.

Each iteration of an iterative image reconstruction algorithm requires many FP operations. The dominant computations are called the "forward projection" and the "back projection" operations; one of each is needed each iteration, and they require about the same amount of computation. The amount of computation (measured in number of FP operations) required for a forward projection operation is approximately

$$4 \cdot N_\mathrm{x} \cdot N_\mathrm{y} \cdot N_\mathrm{z} \cdot N_\mathrm{a} \cdot N_\mathrm{r} \approx 7.2\ \text{trillion FP operations}.$$

An iterative algorithm needs several iterations; the number of iterations depends on the algorithm. The algorithms that are most easily parallelized might need about 100 iterations, each of which needs a foward projection and a back-projection operation. If algorithm advances could reduce the number of iterations to only 10 iterations, then the total operation count would be about $2 \cdot 10 \cdot 7.2 = 144$ trillion FP operations

Portable devices would require fewer helical rotations (N_r) as they would have a narrower region-of-interest. The total number of operations, therefore, would be between 14 and 30 trillion FP operations for 1 or 2 rotations.

2.1 Benchmark Overview

For the purposes of this work, a representative subset of different algorithms used in the reconstruction process were analyzed. Though MRI is a very commonly used imaging technique, generating an MRI image is not considered tomographic image reconstruction as it is not a product of multiple cross-sectional images. This work, however, still presents results using MRI-related benchmarks as they are computationally very similar to the tomographic benchmarks.

MBIR: Model-based iterative reconstruction[26] (MBIR) algorithms work by iteratively minimizing a cost function that captures the physics of an x-ray CT imaging system, the statistics of x-ray CT data, and a priori knowledge about the object (patient) being scanned. By incorporating accurate models, MBIR methods are less sensitive to noise in the data, and can therefore provide equivalent image quality as present-day "filtered back-projection" (FBP) algorithms with much lower x-ray doses. Alternatively, they can provide improved image quality at comparable x-ray doses. Figure 2(c) [26] shows an example of a coronal reformatted slice of a 3D helical x-ray CT scan reconstructed by the conventional FBP method and by an MBIR method. For these thin-slice images, the MBIR method exhibits much lower noise than the FBP images, enabling better diagnoses. The benchmark used here is the most compute-intensive inner-loop in the algorithm.

The Radon Transform: The Radon transform of a continuous two-dimensional function g(x,y) is found by stacking or integrating values of g along slanted lines. Its primary function in computer image processing is the identification of curves with specific shapes.

The Hough Transform: The Hough transform, like the Radon transform, is used to identify specific curves and shapes. It does this by finding object imperfections within a certain type of shape through a voting procedure which is carried out in parameter space. The Hough transform was historically concerned with identifying lines in an image, but has later been used to identify the locations of circles and ellipses.

CT Segmentation: A CT scan involves capturing a composite image from a series of x-ray images taken from various angles around a subject. It produces a very large amount of data that can be manipulated using a variety of techniques to best arrive at a diagnosis. Oftentimes, this involves separating different layers of the captured

Benchmark	#instrs	Data req'd	Registers req'd		#FPU chains		
		$B/instr$	Int.	FP	2-op	3-op	4-op
`MBIR`	17	0.94	12	14	1	0	1
`Radon`	70	0.80	12	18	1	3	2
`Hough`	21	0.95	12	8	1	0	0
`Segment`	86	1.26	16	11	0	6	6
`Laplace`	23	1.04	13	10	1	1	1
`Gauss`	25	0.80	14	9	0	1	0
`MRI.FH`	41	0.88	14	16	1	2	1
`MRI.Q`	37	0.97	14	14	3	0	1

Table 1: Medical imaging application characteristics.

image based on their radio-densities. A common way of accomplishing this is by using a well-known image-processing algorithm known as "image segmentation". In essence, image segmentation allows one to partition a given image into multiple regions based on any of a number of different criteria such as edges, colors, textures, etc. The segmentation algorithm used in this work has three main FP-intensive components, Graph segmenting (`Segment`), Laplacian filtering (`Laplace`), and Gaussian convolution (`Gauss`).

Laplacian Filtering: Laplacian filtering highlights portions of the image that exhibit a rapid change of intensity and is used in the segmentation algorithm for edge detection.

Gaussian Convolution: Gaussian convolution is used to smooth textures in an image to allow for better partitioning of the image into different regions.

MRI Cartesian Scan Vectors: One of the main computational components of reconstructing an MRI image is calculating the value of two different vectors, which are referred to as `MRI.FH` and `MRI.Q` [11, 22] in this paper. These vectors are used to reconstruct an image using non-Cartesian sampling trajectories – a computationally less efficient but faster and less noisy than reconstructing using a Cartesian scan trajectory.

2.2 Benchmark Analysis

Table 1 shows some of the key characteristics of the benchmarks under consideration. The columns are defined as follows: "#instrs" specifies the number of assembly instructions in each of the benchmarks, "Data required" specifies the memory requirements in terms of average number of bytes required per instruction, "Registers required" specifies the number of entries required in integer and FP register files (RFs) so that the benchmark need not spill temporaries to memory, and "#FPU chains" specifies the number of FP computation chains that are 2, 3, and 4 operations deep. From the table, all of these benchmarks are FP-intensive and require a large amount of data for the computation they perform with memory-to-computation ratios ranging from 0.80 to 1.26, well in excess of the 0.15 bytes/instruction supported by the GTX 285 GPGPU mentioned earlier. The loops in these benchmarks are "do-all" loops – there are no inter-iteration dependences. However, each iteration is typically sequential as indicated by the relatively small number of registers that are required. FP computation tends to be organized as moderately deep chains of sequentially dependent computation instructions.

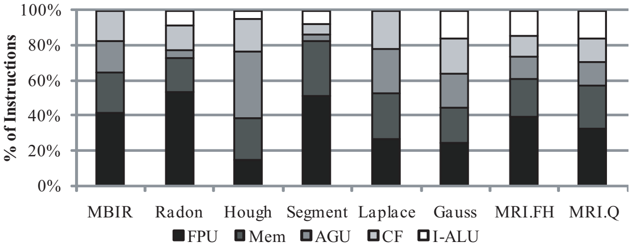


Figure 3: Instruction type breakdown showing the % of instructions used for FP computation (FPU), loads and stores (Mem), address generation (AGU), control-flow (CF) and integer ALU (I-ALU)

Figure 3 characterizes the type and frequency of instructions in each benchmark, showing the percentage of FP arithmetic, memory, address-generation, control-flow, and integer arithmetic instructions, respectively. As can be seen from this graph, the computation in these benchmarks is predominantly FP arithmetic, but there are some integer operations as well. Of the integer registers specified in Table 1, most of these registers are used for memory address generation, as the benchmarks often access elements from multiple arrays and several member variables of data structures. In most cases, the control-flow instructions are those to check the terminating condition of the loop. Benchmarks with a high % of control-flow instructions have if-else conditions within the loop kernel.

3. THE MEDICS PROCESSOR ARCHITECTURE

A computation system for portable medical imaging must meet several requirements in order to deliver high performance while still having low power utilization:

Low-latency FP computation: Most floating-point units (FPUs) have a latency of 3 to 4 cycles. The applications considered here often perform multiple consecutive FP operations on one piece of data before storing the result in memory. Chains of 4 or 5 dependent operations result in execution times of 12 to 15 cycles. Efforts must therefore be made to either reduce the latency of the FPU pipeline or implement low-power techniques to hide this latency.

Wide-SIMD FP pipeline: The algorithms in this domain are all FP-intensive; the representative benchmarks considered in this paper have, on average, 36% of FP instructions in the inner-most, most frequently-executed loops. Further, the loops are all do-all loops – all the iterations of the loops can execute in parallel as there are no inter-iteration dependencies. This property enables simple SIMD-ization of these loops, where subsequent iterations of the loop may be assigned to individial lanes in the SIMD datapath. Further, since SIMD replicates only the arithmetic units, there is comparatively less control overhead in a SIMD design compared to a single-issue design resulting in improved power efficiency. Many of these algorithms have some simple control flow in their inner-most loops, primarily limited to operations such as bounds-checking requiring some support for predication. The ability to broadcast a single value to all SIMD lanes is also required.

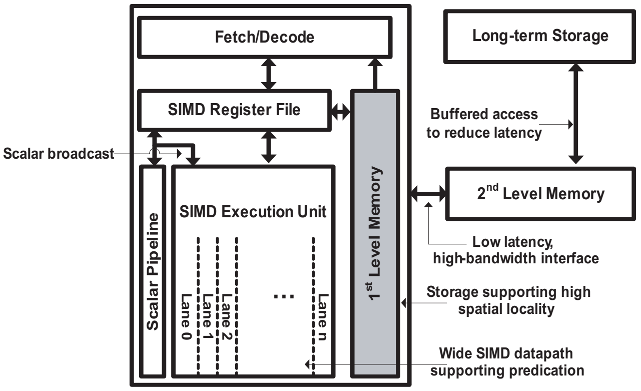


Figure 4: Components required for a medical imaging compute system.

High bandwidth, low-latency storage: Image reconstruction algorithms have a very high memory-to-compute ratios, requiring large amounts of data in a very short time. In order to support this behavior, a sufficient amount of memory has to be available on-chip. In addition to performing little compute per unit data, the total amount of data processed in reconstruction applications is also large. Several gigabytes of storage is therefore required, and should ideally placed as "close" to the processor as possible.

These requirements are combined to create a high-level system sketch for MEDICS shown in Figure 4 and explored in detail below.

3.1 FPU Pipeline

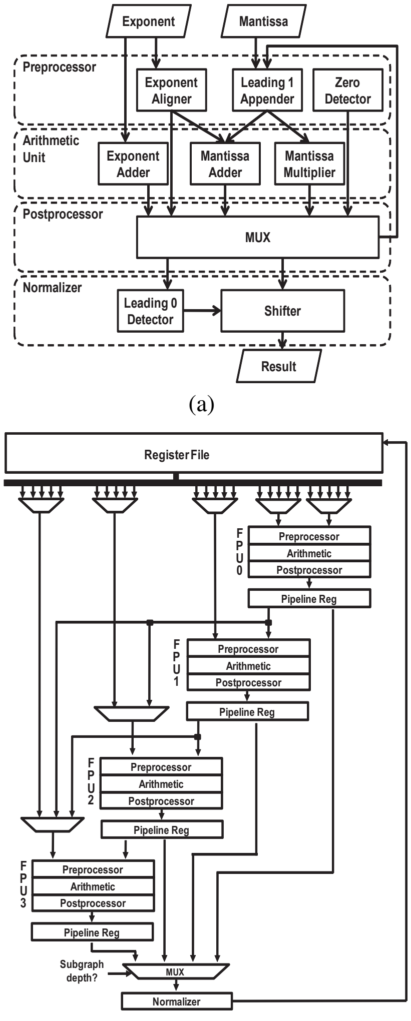
FP operations in reconstruction applications – especially those in frequently executed blocks of code – are FP adds, subtracts and multiplies. While the occassional divide is required, it is usually executed in software with the aid of a reciprocal operation. Efforts are therefore made in this work to optimize the main FPU pipeline, consisting of an FP adder/subtractor and a multiplier.

3.1.1 Reducing FPU Latency

For an FPU with a latency of 3 clock cycles, back-to-back dependent operations may only be issued every 3 cycles resulting in a significant loss in performance. Operation chaining in FPUs helps mitigate this latency, allowing some parallelism in the initial processing and formatting of FP numbers.

In addition to reducing data-dependent stalls, chaining operations has the advantage of reducing the overall number of register-file (RF) accesses; a sequence of two multiply instructions back-to-back, for example, normally requires a total of four read accesses and two write accesses. Chaining will reduce this to three read accesses and a single write access, though it will require an additional read port to perform all the reads simultaneously. The savings from reducing the overall number of accesses is often more than the added cost of a read port.

Reconstruction applications, however, typically have chains of dependent FP operations longer than two operations. This work explores the possibility of extending the principles applied in constructing conventional fused multiply-add FPUs further than is traditionally done allowing for one FP instruction to execute several FP operations in series to best accelerate long FP chains and to reduce RF access power. Further, these units have to be generalized to execute any sequence of FP operations in any order rather than just merely a fused multiply-add/subtract sequence.



(a)

(b)

Figure 5: FPU architecture. (a) Internal structure for FPU. (b) The Normalizer stage may be removed for all but the last in a chain of FPUs

Conventional FPU Architecture: A typical FP adder consists of a zero detector, an exponent aligner, a mantissa adder, a normalizer, and an overflow/underflow detector. The exponent aligner in the FP adder aligns the mantissa for the two operands so as to use the same exponent to compute the addition operation. Meanwhile, an FP multiplier generally consists of a zero detector, an exponent adder, a mantissa multiplier, a normalizer, and an overflow/underflow detector. Following the IEEE-754 standard, FP units also include rounders and flag generators. The rounder takes into account desired rounding modes, namely round to nearest, round toward 0, round toward positive infinity, and round toward negative infinity. The flag generator indicates whether the result is zero, not-a-number, generates an overflow, or generates an underflow. Figure 5(a) illustrates the interaction between these various components.

Chained FPU Design: As depicted in Figure 5(a), the FPU implementation used in this work is divided into four conceptual stages: preprocessor, arithmetic unit, postprocessor and normalizer. Typically, all FPUs operate only on normalized FP numbers

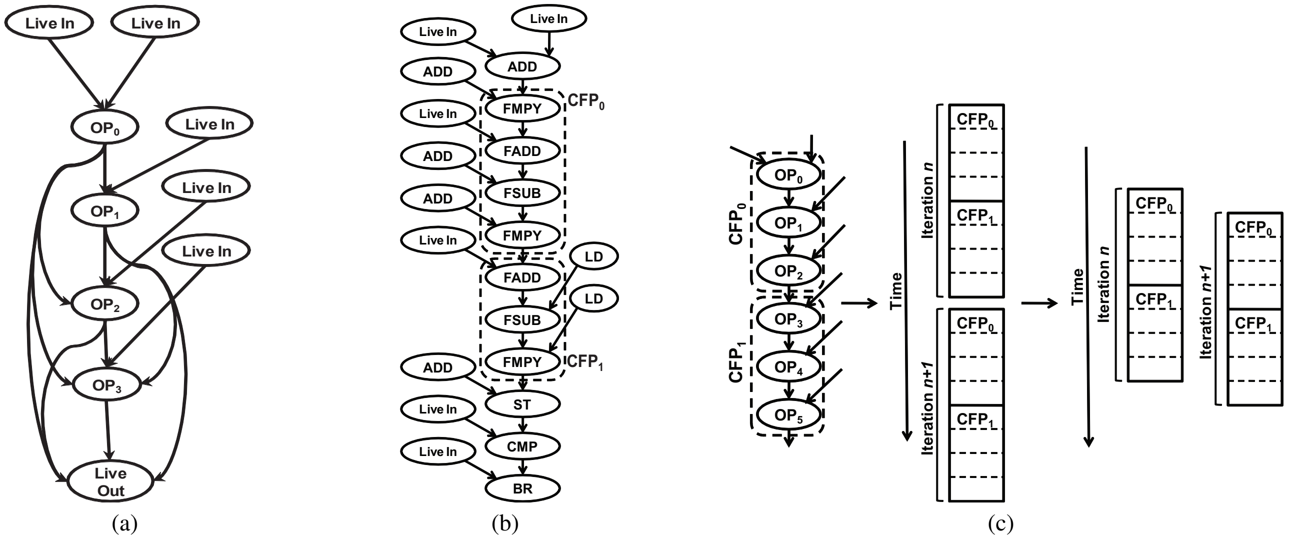


Figure 6: Example using chained FPUs (CFPs) from the Radon benchmark. (a) Operation identification. (b) Latency hiding via software pipelining.

and this is enforced by the normalizer. In general terms, the earlier parts of the FPU pipeline consist of components that expand the IEEE standard-form input operands into intermediate representations suitable for the main arithmetic units, and the later parts of the FPU pipeline compact the results of the computation back into the IEEE representation. When operations are performed back-to-back, the intermediate values are never committed to architected state and, as such, need not be represented in the standard form, saving time on the FPU critical path and reducing the required hardware.

When the normalizer takes the result from the postprocessor, it primarily detects leading zeroes and shifts them as necessary so as to conform to the IEEE-754 FP format. If multiple FPUs are chained together and the value computed in the postprocessor is only an intermediate value, and not one committed to architectural state, the normalizing step may be removed and the next stage in the FPU chain can treat this result as a denormalized value. The normalizer consumes a significant amount of computation time – close to 30% – so its removal results in marked performance improvements. More details about the achievable improvements are presented in Section 4.

Some extra logic that is necessary to make chaining work properly includes the truncation and shifter placed in the postprocessor to process varying result widths, resolve temporary overflows, and detect the true location of leading ones so that they may be correctly appended for the next stage of the computation. The 32-bit representation produced by the postprocessor simplifies the relay between one stage of FP computation to the next, and if deemed necessary, outputs can be pulled out at any FPU to be normalized immediately and its results obtained. The conceptual operation of the chained design is illustrated in Figure 5(b) where the normalization step is performed for only the final operation in the sequence. Additional control logic allows for earlier normalization of intermediate values if a sequence of instructions has fewer than the maximum number of operations. Operations can receive their inputs from either a predecessor or from the register file.

Identifying FP chains: Modifications made to the Trimaran [28] compiler are used to identify and select sequences of instructions for execution on the chained FPU. First, an abstract representation of the possible sequences of execution is created in a data-flow graph (DFG) form. In the DFG, nodes are used to represent each input, output and individual FPUs in the chain. Directed edges are used to represent all possible communication between these nodes. An abstract graph representation for the 4-Op chained FPU in Figure 5(b) is shown in Figure 6(a). This representation has 5 input nodes, 1 output node and 4 operation nodes. Two edges are drawn from the inputs to OP0 and one edge is drawn to one input of OP1, OP2 and OP3. Edges are also drawn from OP0 to OP1, OP2, and OP3; from OP1 to OP2, and OP3; from OP2 to OP3; and, finally, from all of the operation nodes to the output node.

The compiler receives the application source code and this abstract representation as input. It then draws a DFG for the compute-intensive inner-most loop of the benchmark. We the find subgraphs in the application's DFG that are isomorphic to the FPU's DFG, which provides us with the set of operations that can be executed as one instruction in the FPU chain. A greedy algorithm is then used to select the largest of these subgraphs; i.e. a single, 4-long sequence of operations is preferred over two 2-long sequences.

Figure 6(b) shows a subset of the DFG for the Radon transform benchmark's inner-most loop. Two chained FPUs are identified here - one four FP operations in length and the other three FP operations in length. The sub-graphs used in this work were sequences of dependent FP add, subtract, and multiply operations where intermediate values were not live-out of the DFG but were only consumed locally. The internal interconnection is illustrated in Figure 5(b).

3.1.2 Hiding FPU Latency

While the total time taken to execute several back-to-back FP operations may be reduced using FPU chaining, it significantly increases the total pipeline depth and, consequently, the latency of any FP instruction. Traditional architectures use hardware multithreading to hide various sources of latency – control latency, computation latency, memory latency, etc. While hardware multithreading helps increase the overall performance, it has a few drawbacks. Firstly, the control when using multithreading is significantly more complicated as each thread has its own PC, machine status register, execution trace, etc. In addition, each thread must be presented with the same architectural state. This work takes a compiler-directed approach of hiding the long FPU pipeline latency by software pipelining the inner-most loops [16] and overlapping independent successive iterations as shown in Figure 6(c). When using software pipelining, since more data is being processed in parallel, the RF must be increased in size to provide enough data for the required computation. Instantaneous power will also increase due to the increase in operations at any given time, but the overall energy of computation will decrease since the processor is spending more time doing useful work rather than idling and waiting to issue a new instruction.

3.2 Data Compression

3.2.1 *Lossy Compression*

One approach to narrow the bandwidth gap and make more effective use of the computing resources available is to reduce the total amount of data required by the application. Iterative CT reconstruction techniques [6, 26] are inherently error-tolerant since they keep iterating until the number of artifacts in the image is within some tolerance. A technique used in MEDICS that exploits this in a power-reducing manner is the use of 16-bit floating-point computation mentioned in the IEEE 754-2008 (or IEEE 754r) standard. These "half-precision" floating-point values consist of a sign bit, a 5-bit exponent field and a 10-bit mantissa field as opposed to the 8-bit exponent field and 23-bit mantissa field used in the 32-bit standard.

The advantages of changing the floating-point width are twofold. The first advantage is an effective doubling of bandwidth; i.e. twice as many operations can now execute using the same bandwidth as before. The authors of [10] explored the effects of using 16-bit floating point on CT image reconstruction. Not only did [10] conclude that the error produced by this reduced precision was well within tolerance but that the resulting increase in bandwidth has shown over a 50% increase in image reconstruction performance on Intel CPUs.

The second advantage is a reduction in the datapath hardware; our experiments showed a 58.5% reduction for a 2-input FPU. Further, the FP register file size can also be reduced to a 16-bit width. The effects of this change are discussed in section 4.

3.2.2 *Lossless Compression*

Several recent studies have demonstrated the effectiveness of compression in reducing the size of CT scan results. These studies have primarily been focused on long-term storage size, but similar principles are applied in this work to improve performance. In [1], the Lempel-Ziv and Huffman lossless compression algorithms are used to compress sinogram data from PET scans. In a similar manner, the JPEG-LS lossless compression algorithms are applied to compress sinogram data from CT scans [2]. They explore using lossy compression algorithms as well and demonstrate that compression ratios up to 20:1 are possible with minimal artifacts in the final reconstructed image. Lossless compression of sinograms results in 10:1 and greater compression ratios. An important point to note is that the compression ratios when lossless techniques are applied to final images are considerably less – on the order of 2:1 and 3:1. Sinograms are generally much more compressible than images due to their highly correlated structure.

Further, rather than using software techniques for compression, this work takes the approach of using ASICs for performing compression and decompression. Using software-based compression provides greater flexibility in terms of the compression algorithm used and at what granularity data has to be compressed. However, it leads to a considerable loss of performance, up to 50% [20] in some cases, without accounting for the fact that the main application cannot execute on the core while the compression and decompression software is running. For this reason, this work focuses on a hardware-only approach to compression using ASIC implementations of the JPEG-LS compression algorithm [14, 15, 20]. For a 32-bit pixel, this hardware implementation has an area footprint of 0.07mm^2, power consumption of 1.6mW, and can operate at a frequency of 500 MHz.

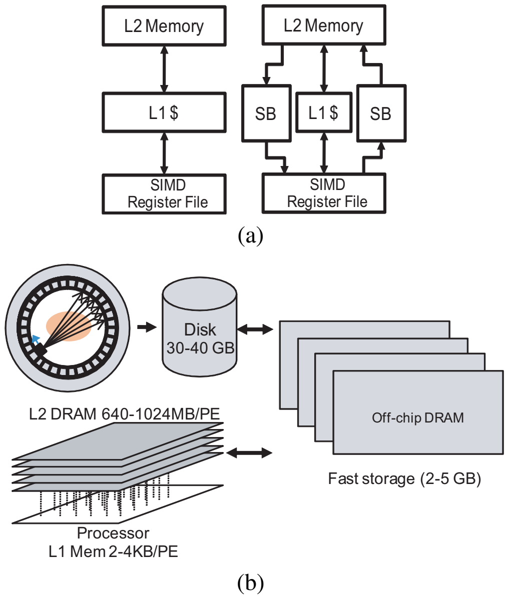


Figure 7: Memory Interfaces. (a)Datapath-to-DRAM memory system. A large L1 or a smaller L1 with input and output streaming FIFOs may be used. (b)Off-chip memory system

3.3 Memory System

In this work, a 3D die-stacked DRAM array is used for high-density, low latency storage. Where traditional designs incur a latency of several hundred cycles while accessing on-board DRAM storage, utilizing 3D stacking provides the storage density of D-RAM but without incurring the wire delay penalty of going off-chip. The 3D DRAM cells are accessed using through-silicon vias (TSVs). TSVs are essentially identical to the vias normally used to connect adjacent metal layers in silicon [9]. The specific implementation used in this paper is that provided by Tezzaron Corporation [25]. This implementation, at a 130nm technology, has a density of 10.6 MB/layer-per-mm^2, a DRAM-to-core latency of 10ns, a data throughput of 4 giga-transfers/sec, and supports 10,000 TSVs each mm^2; i.e., for a memory interface of area 0.1 mm^2, the DRAM-to-core memory bandwidth supported is up to 500 GB/s, considerably higher than any off-chip memory bandwidth currently offered, and much beyond the bandwidth requirements of most applications, including that of medical image reconstruction. Further, when deployed on a core running at 500 MHz, the 10ns delay results in a total latency of approximately 10 cycles to send the required memory address to the DRAM and receive the appropriate data. This latency is comparable, or faster, to modern L2 caches and, for this reason, the 3D-DRAM storage will henceforth be known as "L2 memory" in this paper.

3.3.1 *On-chip Memory Organization*

Two potential on-chip memory systems are considered, as depicted in Figure 7(a). The first, more traditional system, is to use an L1 SRAM cache between the SIMD RF and the L2 memory in

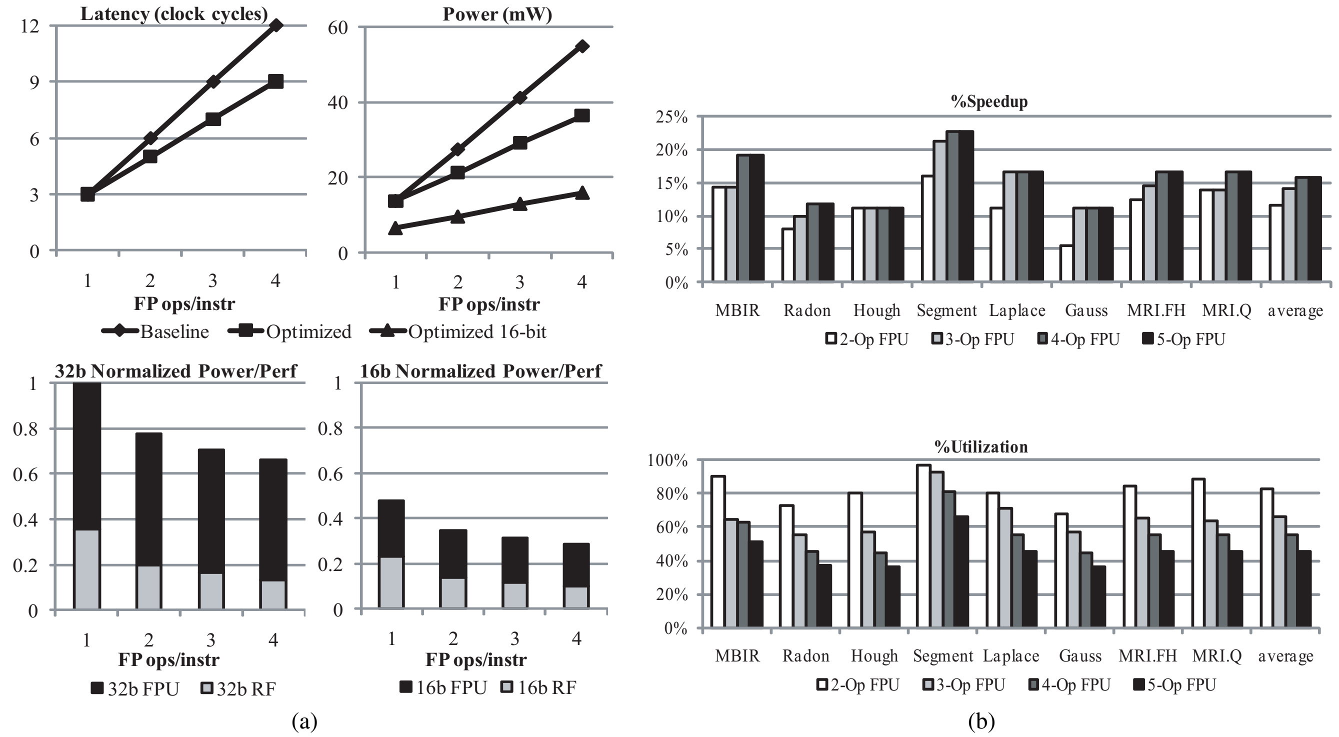


Figure 8: (a) Datapath latency, power and area effects of varying the number of FPUs when increasing FPU chain length. (b) Speedup and FPU utilization with increasing chain length

order to hide the latency of an L2 memory access. The second system takes advantage of the streaming nature of these benchmarks by employing "streaming buffers" (SBs) to continuously fetch and store data at pre-specified offsets. These are two alternatives to the approach used in modern GPGPUs – utilizing a large number of thread contexts and some caching to keep the computation units busy while data is fetched from memory. All three techniques, as they apply to this domain, are explored in Section 4.

3.3.2 *Off-Chip Memory Organization*

Sinogram data generated in reconstruction systems is first stored in a "recon box" hard disk drive. The data on the drive is then processed and reconstructed using whichever processor is used in the machine. A similar design is envisioned for the system proposed in this work as shown in Figure 7(b). Additional steps are taken, however to further hide the long latency of a disk access. This is done by using a very large array of off-chip DRAM (2 to 5 GB). The on-board storage size selected is an appropriate match for repeated access to the large working set size required by advanced reconstruction algorithms, as mentioned in Section 2.

4. EXPERIMENTAL EVALUATION

The major components of MEDICS were designed in Verilog and synthesized at 500 MHz on a 65nm process technology using the Synopsys Design Compiler and Physical Compiler. Power results were obtained via VCS and Primetime-PX, assuming 100% utilization. Area and power characteristics for regular memory structures like dual-ported RFs and caches were obtained through a 65nm Artisan memory compiler while RFs with more than 2 read ports were designed in Verilog and synthesized. The benchmarks were SIMD-ized by hand and compiled using the Trimaran compiler infrastructure [28]. A SIMD width of 64 was chosen as having a wider datapath led to the scalar broadcast interconnect being on the critical path and reducing the efficiency of the processor.

4.1 FPU Chaining

Table 1 indicates that a number of the applications in this domain have several long sequences of back-to-back FP operations. Based on this data, the FP datapath in MEDICS was designed with an FPU consisting of 4 back-to-back operations. Figure 8 shows the effects of varying the number of operations in the FPU.

In Figure 8(a), the x-axis for all the graphs, "FP ops/instruction" is the number of successive, dependent FP operations executed in the chained FPU. The "latency" graph shows the time (in 2ns clock cycles) taken to execute an input subgraph. The baseline 3-cycle FPU takes 3 cycles for each operation and thus has a latency that increases by 3 cycles for every added operation. The removal of redundant hardware in the optimized FPU chain results in significantly less overall latency – a savings of 3 cycles when 4 FPUs are connected back-to-back.

The "power" graph illustrates the power savings obtained from optimized normalizing. Here, the baseline is multiple un-modified FPUs executing operations back-to-back. In this graph, too, the gap between optimized and unoptimized FPUs widens quite dramatically as the number of operations per instruction increases. This gap widens further when the width of the datapath is reduced to 16 bits as mentioned in Section 3.2.1. The normalizing hardware removed as part of the FPU optimization process removes a higher percentage of the hardware in a 16-bit FPU than in a 32-bit FPU, resulting in reduced baseline and incremental power with each added FPU in the chain. The power measurement in this graph is the sum of the RF access power and the FPU execution power to better reflect the power penalty from increasing the number of RF read-ports.

The "power/perf" graphs address the efficiency of the different solutions. They show the normalized power consumed per operation to achieve an IPC of 1; i.e., with every stage of the FPU occupied and busy. Here, too, the power consumed for the amount of work done steadily reduces as the number of FP operations per instruction increases. While the access power for an RF increases for every added read port, the improvement in the efficiency of the RF shown in the graph indicates that this is amortized by the reduction in overall accesses and by the performance improvement achieved by chaining together FPUs.

Figure 8(b) shows the speedup and FPU utilization observed for the different benchmarks when using 2-, 3-, 4-, and 5-operation chained FPU datapaths. The speedup varies based on how frequently the various chained FP operations occur in each benchmark (see Table 1) and the latency penalty incurred when issuing back-to-back dependent FP operations. The benchmarks that had the most to gain from chaining FPUs were the ones with the most number of 4-long chains of FP operations – `MBIR` and `Segment`, for example. On average, 12%, 14% and 16% speedups are observed when using 2-op, 3-op and 4-op chained FPUs, respectively. Speedup saturates at 4 operations and adding a fifth operation in the chain only reduces the overall utilization of the FPU. Using a 4-op chain is, therefore, the best solution.

4.2 Local Storage

Two aspects of local storage were evaluated. The first is the number of entries in the main RF. A baseline RF size of 16 elements was chosen as this captures the register storage requirement for the majority of the benchmarks shown in Table 1. The larger an RF, the more power it consumes per access, and so if only the instantaneous power consumption is considered, using a smaller RF is the preferred solution. However, this does not account for the increase in the application run-time associated with spill instructions inserted due to a lack of available registers. Therefore, the metric chosen to evaluate the efficiency of various RF sizes was the energy consumed by individual iterations of the applications, accounting for RF access and execution energy. Figure 9(a) shows the energy consumption when using a 32-bit 8-entry RF and a 32-bit 32-entry RF, normalized to that of a 16-entry RF. The energy consumed when using an 8-entry RF is significantly higher than 1 – almost double in the case of the `Segment` benchmark – and is only less than 1 for the `Hough` benchmark, which requires only 8 registers. The energy overhead of using a 32-entry RF is quite small – less so for the `Radon` benchmark since it actually requires 18 registers to not have any spill code. Therefore, while a 16-entry RF is chosen for the MEDICS design, the added power and energy overhead of increasing this to a 32-entry RF is quite minimal.

The second aspect of local storage that was evaluated was the mechanism used to hide the latency of the L2 memory. Three different solutions were analyzed using an L1 cache, using streaming FIFO buffers, and replicating register contexts. To hide 10 cycles of L2 memory latency, sufficient buffering is required for 10 memory instructions, or 40 bytes for 32-bit data values. This number is increased to the nearest power of 2 to 64 bytes per SIMD lane. This additional storage may be used for miscellaneous data like register spill.

An L1 cache of 64 bytes/lane can be used, but this does not effectively exploit the spatial locality present in these applications. Since all of these algorithms process images sequentially, pixel-by-pixel, a 64 bytes/lane streaming data buffer FIFO and DMA engine coupled together to transfer multiple loop iterations worth of data in response to a single request is a better solution. This process reduces execution time since explicit address computation for loads and stores may be eliminated from the main datapath. There is added performance overhead of programming the FIFO, but this is amortized over the length of the loop. A small L1 cache of 16 bytes would still required for miscellaneous data as mentioned earlier. Another technique, used in modern GPGPUs, is to hide memory latency by simply using thousands of register contexts.

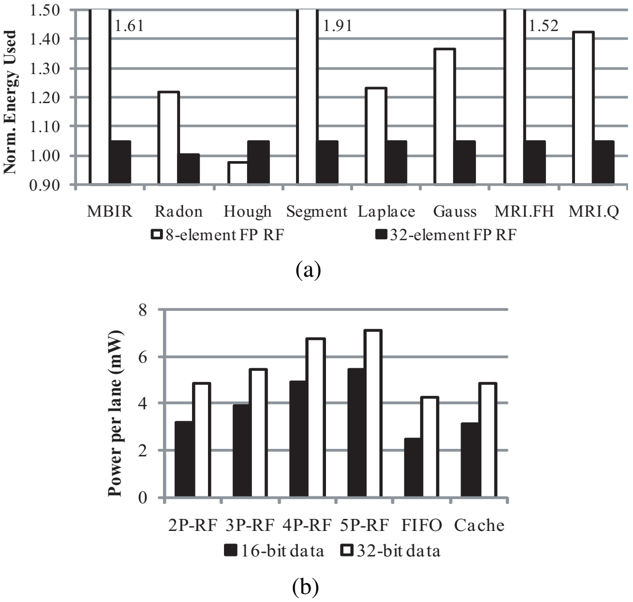


Figure 9: Local storage characteristics. (a) Energy consumption when using 8-entry and 32-entry RFs, normalized to energy consumption of 16-entry RF. (b) Power consumption of each additional RF context, using a FIFO streaming buffer and using an L1 cache to hide L2 memory latency. The FIFO and context-based solutions include an additional 16-byte/lane L1 cache for register spill.

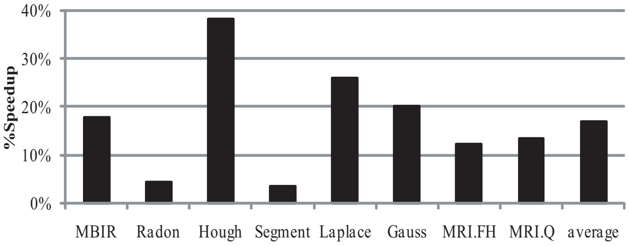


Figure 10: Speedup using a FIFO streaming buffer instead of an L1 cache.

The power consumptions of these different techniques was evaluated, with both 16-bit and 32-bit datapaths, and the results are shown in Figures 9(b) (here, "*n*P-RF" is the power overhead per context when utilizing 16-entry RFs with *n* read ports). This power is multiplied for each context required, i.e. if a processor has 10,000 16-entry contexts, the power consumed by the dual-ported register files will be approximately 5 Watts. Due to its high power consumption, the common technique of having several parallel contexts is the least economical. This is evident even though the L2 memory latency that has to be hidden is very low, compared to that of the off-chip DRAMs used in modern GPGPUs. The overhead naturally increases as read ports are added to RFs to support FPU chaining. The cache consumes approximately the same amount of power as an additional dual-ported RF. The streaming FIFO buffers, though, consume less power than either solution, primar-

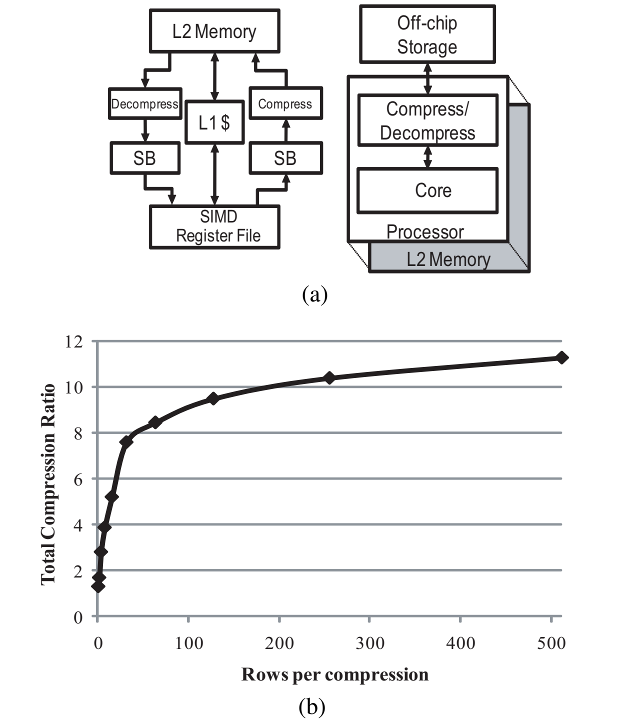


Figure 11: Image compression. (a) Datapath-to-DRAM memory system with two different possible configurations for using compression and decompression engines. (b) Degrading compression ratios when increasing the granularity of loss-less compression.

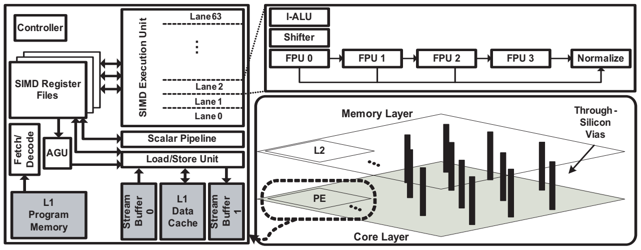


Figure 12: MEDICS processor architecture. The figure on the bottom-right conceptually illustrates the design of the entire chip. The figure on the left shows the architecture of an individual PE. The figure on the top-right shows the arithmetic execution pipeline.

ily due to the lack of any addressability. The case for the streaming buffers is further underscored in Figure 10 which illustrates the speedup achieved from not having to always issue address-calculating instructions when using the streaming buffers – 17%, on average.

Based on the performance improvement observed and their low power consumption, FIFO buffers and a small L1 cache provide the best interface between the datapath and L2 memory. It is not necessary to add hardware to maintain coherence between these structures; the primary input and output data structures of the applications considered are accessed through the FIFO buffers and the L1 is only used for storing any intermediate values such as register spill. Therefore, as there is no overlap between data stored in the L1 and data stored in the FIFOs, area and power-expensive hardware coherence schemes need not be used in this design.

4.3 Compression

Two options for integrating hardware compression were considered: in-pipeline and compression at the memory bus as shown in Figure 11(a). In the first approach, the compression and decompression engines are placed directly in the main datapath. In this scheme, compressed data is stored on the disk, on the on-board, off-chip DRAM, and in the L2 memory. Compression and decompression hardware is placed between the L2 memory and the streaming buffers. The DMA engine responsible for filling and clearing the streaming buffer FIFOs triggers the decompression and compression, respectively. The decompressed data, however, must fit in the streaming buffer FIFO which is only a few bytes in size per lane. Considering each row is 2 kB in size, a small subset or fraction of rows in each sinogram must be compressed together rather than compressing an entire 512-row sinogram.

In the second approach, only the disk and on-board storage hold compressed data and data is decompressed before it is stored in the L2 memory. The main advantage with this approach is the improvement in compression ratios. Figure 11(b) shows the typical degradation of compression ratios as fewer and fewer rows of a sinogram are compressed. If an entire slice (512 rows) is compressed, its size shrinks from 1.05 MB to 93.2 kB (11.3:1 compression), whereas if the slice is compressed row-by-row, its overall size is 1.02 MB (1.3:1 compression) – a negligible reduction in size. Due to the improved compression achieved by decompressing data into the L2 memory, MEDICS's positioning of the compression/decompression engines on the boundary of on-chip and off-chip storage is the better of the two techniques.

Using compression engines on the memory bus interface allows increasing the total number of PEs on the processor, thereby increasing the total processing capability. The decompression engines used have a throughput of 1 pixel/cycle, resulting in a latency of 2^{18} cycles per 512x512 image. At this rate, using a 64-wide SIMD PE, at least 64 instructions need to execute per 4-byte pixel generated in order to not be limited by the decompression engines. While this is feasible for the benchmarks with lower memory footprints, the ones with higher memory footprints require almost eight times this throughput, or 8 decompression engines running in parallel, processing different chunks of data.

5. THE MEDICS SYSTEM

The MEDICS architecture is shown in Figure 12. The 3D-stacked DRAM interface is illustrated in the lower-right part of the figure. An individual PE is illustrated on the left, showing the separate scalar and vector pipelines. The top-right shows an individual lane of the vector pipeline in more detail. A compound FPU which does 4 FP operations back to back is used in this design. The RF has 16 elements for each of the FP and integer RFs based on the data in Table 1 and FIFO stream buffers are used to transfer data between the datapath and the L2 memory.

The MEDICS processor's design characteristics and power consumption breakdown are shown in Figure 13. Figure 13(a) shows the specifications of each individual PE. The 16-bit version consumes significantly less power than the 32-bit version. However, due to the reduced size of the FP datapath, the area is also reduced, leading to approximately 11% less on-chip stacked DRAM. This is not a problem, though, since the reduced bitwidth effectively leads to a doubleing of the DRAM's utilization. Figure 13(b) shows a component-by-component breakdown of the power consumed in the MEDICS processor. The most significant power reduction from

	16-bit	32-bit
Frequency	500 MHz	
SIMD Lanes	64	
Peak Performance	128 GFLOPs	
Peak Total Power	1.58W	3.05W
Total Area	36.3 mm^2	40.7 mm^2
On-chip DRAM	774 MB	867 MB
Efficiency	81.1 Mops/mW	42.9 Mops/mW

(a)

Component	16-bit Power	32-bit Power
4-op 16-bit FPU	10.30 mW/ln.	29.15 mW/ln.
16-element 16-bit RF	5.48 mW/ln.	7.14 mW/ln.
Local stream buffers	2.51 mW/ln.	4.25 mW/ln.
Etc. datapath (scalar pipe, AGU, control)	59 mW	
DRAM Power	350 mW	390 mW

(b)

Figure 13: MEDICS specifications. (a) Overall per-PE specifications (b) Power breakdown of individual components

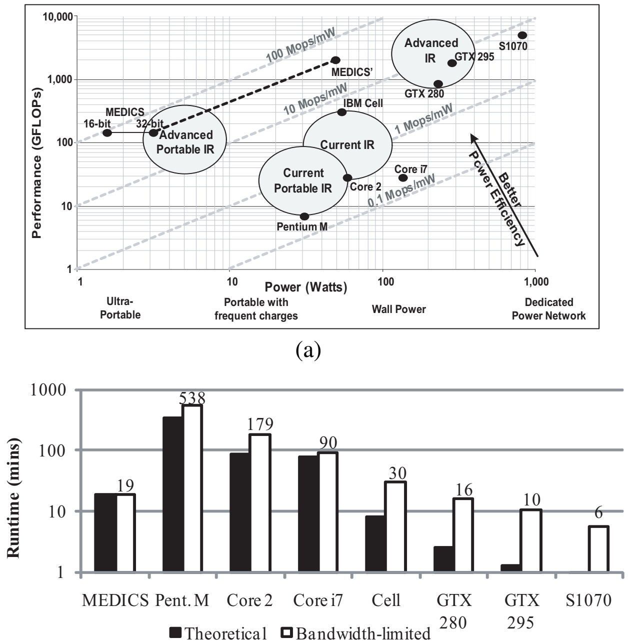
the 32-bit to the 16-bit datapath is seen in the 4-op, 5-input FPU which sees a 64% reduction.

In addition are the area and power of the 8 sets of compression/decompression engines required for sufficient throughput, the power consumption of the processor changes very little with this addition and the area increases by 1.2mm^2. Using this compression mechanism, the effective bandwidth seen by the processor is 10X the nominal, allowing for an equivalent increase in the amount of processing power and improved scalability while maintaining performance/power efficiency.

Figure 14(a) shows a modified Figure 1. Extra data points have been added to show how MEDICS compares in performance/power efficiency to the other processors in consideration. The 32-bit MEDICS system has an efficiency improvement of 10.6X over the Nvidia GTX 280 and 6.8X over the Nvidia GTX 295, which are also fabricated on a 65nm process. The 16-bit MEDICS system has a 20.5X and 13.1X improvement, respectively. Its performance and power consumption makes it an excellent choice for advanced, low-power image reconstruction. If more PEs are used, and the necessary compression engines and on-chip communication network are added, the design scales to the denoted as MEDICS' on the plot. At this point, it delivers the *same* performance as the GTX 295 while consuming significantly less power.

For an off-chip bandwidth of 141 GB/s (the same as that of Nvidia GTX 280), MEDICS has a peak data consumption of 1.11 bytes/instr. Given the benchmarks in consideration, the only benchmark that would be bandwidth-limited is the `Segment` benchmark, which requires 1.26 bytes/instr. The peak consumption of the other processors in consideration, however, ranges between 0.08 bytes/instr for S1070 to 0.85 bytes/instr for the Core i7, all lower than what is required for this domain. The ramification of this disparity is best illustrated in Figure 14(b). This graph shows the overall run-time of the MBIR reconstruction application [26], assuming that the application requires the full 144 trillion operations specified in Section 2. The "theoretical" bar shows what the run-time would be if the listed peak performance rating were actually possible. The "realized" bar shows what is actually possible given the bandwidth constraints of the various processors under consideration.

While consuming 1 to 2 orders of magnitude less power than all the other existing solutions, MEDICS delivers reconstruction run-times that are matched only by high-end desktop GPGPUs. It is only significantly outperformed by the server-class S1070 which consumes over 100X the power while only reducing the run-time by two-thirds. The principle efficiency improvements come from:



(a)

(b)

Figure 14: (a) *(Modified Fig. 1)* Suitability of MEDICS for the performance and power characteristics of the domain (b) Theoretical and realized (bandwidth-limited) run-times for advanced reconstruction algorithms

- Optimized FPU design
- Fewer hardware contexts
- Increased bandwidth to off-core storage
- Improved on-chip latency-hiding
- Removing power-hungry application-specific hardware (e.g. texture units)

6. RELATED WORK

There are multiple current hardware solutions for medical image reconstruction, based on DSPs [24], general-purpose processors [27] and GPGPUs [12]. These are unsuitable for the next generation of low-power image reconstruction systems for the reasons enumerated in Section 1, such as lack of floating-point supprt, insufficient performance and high power consumption.

There are several other examples of low-power, high-throughput SIMD architectures like SODA [30] for signal-processing but it, being a purely integer architecture, is unsuitable for this domain space. There are also high-throughput floating point architectures such as TRIPS [19], RAW [23] and Rigel [8], but these are more focused on general-purpose computing and do not have the same power efficiency as MEDICS, nor do they address any bandwidth concerns. There has also been recent work on image-processing [4, 11] and physics simulations [31] targetting domains traditionally addressed by commercial GPGPUs but these, too, do not address bandwidth and power to the extent that this work does.

Some CRAY systems use a "chained FPU" design. However, these are essentially just forwarding paths across different SIMD lanes. While this connectivity reduces register acceses, the FPU itself was not redesigned the way the MEDICS FPU was.

Other prior work specifically targetting medical imaging has predominantly focused on using porting existing programs to the commercial products mentioned earlier. For instance, in [7], the authors port "large-scale, biomedical image analysis" applications to multi-core CPUs and GPUs, and compare different implementation strategies with each other. In [18], the authors study image registration and segmentation and accelerate those applications by using CUDA on a GPGPU. In [22], the authors use both the hardware parallelism and the special function units available on an Nvidia GPGPU to dramatically improve the performance of an advanced MRI reconstruction algorithm.

7. CONCLUSION

The MEDICS architecture is a power-efficient system designed for efficient medical image reconstruction. It consists of PEs of wide SIMD floating-point engines designed around the computation requirements of the image reconstruction domain. Each PE achieves a high performance-per-power efficiency by using techniques such as FPU-chaining, streaming buffers and compression hardware. As applications in this domain are normally executed on high-performance, general-purpose processors and GPGPUs, these architectures were used to gauge the performance and efficiency of MEDICS. The results are very encouraging, with MEDICS achieving over 20X the power efficiency. The design is also bandwidth-balanced so that all of the performance available on the processor may be effectively used for computation.

Acknowledgements

We thank the anonymous referees who provided excellent feedback. We would also like to think Prof. Jeffrey Fessler for providing valuable insight into the medical imaging domain. This research was supported by National Science Foundation grant CNS-0964478 and ARM Ltd.

8. REFERENCES

[1] E. Asma, D. Shattuck, and R. Leahy. Lossless compression of dynamic PET data. *IEEE Transactions on Nuclear Science*, 50(1):9–16, Feb. 2003.

[2] K. Bae and B. Whiting. CT data storage reduction by means of compressing projection data instead of images: Feasibility study. *Radiology*, 219(3):850–855, June 2001.

[3] A. B. de Gonzalez et al. Projected cancer risks from computed tomographic scans performed in the United States in 2007. *Archives of Internal Medicine*, 169(22):2071–2077, 2009.

[4] V. Govindaraju et al. Toward a multicore architecture for real-time ray-tracing. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 176–197, Dec. 2008.

[5] T. Gunnarsson et al. Mobile computerized tomography scanning in the neurosurgery intensive care unit: increase in patient safety and reduction of staff workload. *Journal of Neurosurgery*, 93(3):432–436, Sept. 2000.

[6] A. K. Hara et al. Iterative reconstruction technique for reducing body radiation dose at CT: Feasibility study. *American Journal of Roentgenology*, 193(3):764–771, Sept. 2009.

[7] T. D. Hartley et al. Biomedical image analysis on a cooperative cluster of GPUs and multicores. In *Proc. of the 2008 International Conference on Supercomputing*, pages 15–25, 2008.

[8] J. Kelm et al. Rigel: An architecture and scalable programming interface for a 1000-core accelerator. In *Proc. of the 36th Annual International Symposium on Computer Architecture*, pages 140–151, June 2009.

[9] G. H. Loh. 3D-Stacked memory architectures for multi-core processors. In *Proc. of the 35th Annual International Symposium on Computer Architecture*, pages 453–464, 2008.

[10] C. Maass et al. CT image reconstruction with half precision floating point values, 2009.

[11] A. Mahesri et al. Tradeoffs in designing accelerator architectures for visual computing. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 164–175, Nov. 2008.

[12] NVIDIA. GeForce GTX 200 GPU architectural overview, 2008. http://www.nvidia.com/docs/IO/55506/GeForce_GTX_200_GPU_Technical_Brief.pdf.

[13] J. Orchard and J. T. Yeow. Toward a flexible and portable CT scanner. In *Medical Image Computing and Computer-Assisted Intervention 2008*, pages 188–195, 2008.

[14] M. Papadonikolakis et al. Efficient high-performance ASIC implementation of JPEG-LS encoder. In *Proc. of the 2007 Design, Automation and Test in Europe*, pages 159–164, Apr. 2007.

[15] M. Papadonikolakis et al. Efficient high-performance implementation of JPEG-LS encoder. *Journal of Real-Time Image Processing*, 3(4):303–310, Dec. 2008.

[16] B. R. Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proc. of the 27th Annual International Symposium on Microarchitecture*, pages 63–74, Nov. 1994.

[17] M. T. Review. Cheap, Portable MRI, 2006. http://www.technologyreview.com/computing/17499/?a=f.

[18] A. Ruiz, M. Ujaldon, L. Cooper, and K. Huang. Non-rigid registration for large sets of microscopic images on graphics processors. *Springer Journal of Signal Processing*, May 2008.

[19] K. Sankaralingam et al. Exploiting ILP, TLP, and DLP using polymorphism in the TRIPS architecture. In *Proc. of the 30th Annual International Symposium on Computer Architecture*, pages 422–433, June 2003.

[20] A. Savakis and M. Piorun. Benchmarking and hardware implementation of JPEG-LS. In *2002 International Conference on Image Processing*, pages 949–952, Sept. 2002.

[21] N. Sinha and J. Yeow. Carbon nanotubes for biomedical applications. *IEEE Transactions on Nanobioscience*, 4(2):180–195, June 2005.

[22] S. S. Stone et al. Accelerating advanced MRI reconstructions on GPUs. In *2008 Symposium on Computing Frontiers*, pages 261–272, 2008.

[23] M. B. Taylor et al. The Raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 22(2):25–35, 2002.

[24] Texas Instruments. CT Scanner: Medical Imaging Solutions from Texas Instruments, 2009. http://focus.ti.com/docs/solution/folders/print/529.html.

[25] Tezzaron Semiconductor. FaStack Memory, 2009. http://www.tezzaron.com/memory/FaStack_memory.html.

[26] J.-B. Thibault, K. Sauer, C. Bouman, and J. Hsieh. A three-dimensional statistical approach to improved image quality for multi-slice helical CT. *Medical Physics*, 34(11):4526–4544, Nov. 2007.

[27] Trenton Systems. Trenton JXT6966 News Release, 2010. http://www.trentontechnology.com/.

[28] Trimaran. An infrastructure for research in ILP, 2000. http://www.trimaran.org/.

[29] D. Weinreb and J. Stahl. The introduction of a portable head/neck CT scanner may be associated with an 86% increase in the predicted percent of acute stroke patients treatable with thrombolytic therapy, 2008. Radiological Society of North America.

[30] M. Woh et al. From SODA to scotch: The evolution of a wireless baseband processor. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 152–163, Nov. 2008.

[31] T. Yeh et al. ParallAX: an architecture for real-time physics. In *Proc. of the 34th Annual International Symposium on Computer Architecture*, pages 232–243, June 2007.

[32] J. Zhang et al. Stationary scanning x-ray source based on carbon nanotube field emitters. *Applied Physics Letters*, 86(18):184104, 2005.