Parallelization and Performance of 3D Ultrasound Imaging Beamforming Algorithms on Modern Clusters

F. Zhang, A. Bilas, A. Dhanantwari*, K.N. Plataniotis, R. Abiprojo*, and S. Stergiopoulos*

Dept. of Electrical and Computer Engineering,
10 King's College Road, University of Toronto,
Toronto, Ontario, M5S3G4, Canada

*Defense R&D Canada,Toronto
1133 Sheppard Ave. West,
North York, Ontario, M3M3B9, Canada

{fanzhang, bilas}@eecg.toronto.edu, kostas@dsp.toronto.edu, {amar.adhanant, robert.abiprojo, stergios.stergiopoulos}@drdc-rddc.gc.ca

ABSTRACT

Recently there has been a lot of interest in improving the infrastructure used in medical applications. In particular, there is renewed interest on non-invasive, high-resolution diagnostic methods. One such method is digital, 3D ultrasound medical imaging. Current state-of-the-art ultrasound systems use specialized hardware for performing advanced processing of input data to improve the quality of the generated images. Such systems are limited in their capabilities by the underlying computing architecture and they tend to be expensive due to the specialized nature of the solutions they employ.

Our goal in this work is twofold: (i) To understand the behavior of this class of emerging medical applications in order to provide an efficient parallel implementation and (ii) to introduce a new benchmark for parallel computer architectures from a novel and important class of applications. We address the limitations faced by modern ultrasound systems by investigating how all processing required by advanced beamforming algorithms can be performed on modern clusters of high-end PCs connected with low-latency, high-bandwidth system area networks. We investigate the computational characteristics of a state-of-the-art algorithm and demonstrate that today's commodity architectures are capable of providing almost-real-time performance without compromising image quality significantly.

Keywords

Parallel processing, Medical applications

Categories and Subject Descriptors

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

ICS'02, June 22-26, 2002, New York, New York, USA.

General Terms

Algorithms, Performance

1. INTRODUCTION

Major efforts have been devoted recently in improving non-invasive, high-precision diagnostic methods, a critical component in the renewed effort to enhance health services. One of the research directions taken to address such requirements is the development of high-resolution, digital, three dimensional (3D) ultrasound medical imaging systems. With the advent of high performance computing facilities and the availability of transducer crystal technology, ultrasound imaging systems have emerged as efficient methods to extract and reproduce relevant medical diagnostic information.

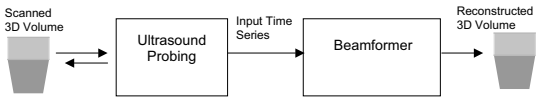

Figure 1: The components of a beamforming-based ultrasound system.

Generally, a digital ultrasound system consists of a set of sensors [25, 11, 2] that perform data acquisition and a back-end computing architecture, responsible for processing the raw data and reconstructing the ultrasonic images [24]. Figure 1 depicts a typical ultrasound system. The ultrasound probing apparatus consists of a set of sensors and a data acquisition unit that probe the object under consideration and gather the sampled data. The beamformer is based on some computing structure and performs signal processing of the samples in order to reconstruct the image of the scanned object. In this framework, both the probing unit as well as the computing architecture components are important in delivering good quality diagnostic results.

Current limitations in sensor technologies necessitate the usage of complex signal processing engines to improve image quality. Most ultrasound medical systems suffer from poor image resolution. Some of these limitations can be attributed to fundamental physical aspects of the ultrasound transducer and the interaction with the tissue. Advanced signal processing algorithms can enhance image resolution and detection quality as well as minimize the relevant prob-

ing hardware requirements leading to cost effective ultrasound system technologies. However, the current state-of-the-art in high-resolution, digital, 3D ultrasound medical imaging faces two main challenges.

First, the ultrasound signal processing structures used are computationally demanding. Traditionally, specialized computing architectures and hardware have been used to provide the levels of performance *and* I/O throughput required, resulting in high system design and ownership costs. With the emergence of high-end workstations and low-latency, high-bandwidth interconnects [10, 3, 6], it now becomes interesting and timely to investigate if such technologies can be used in building low-cost, high-resolution, 3D ultrasound medical imaging systems.

Second, although beamforming algorithms have been studied in the context of other applications [4], little is known about their computational characteristics with respect to ultrasound-related processing, and medical applications in general. It is not clear which parts of these algorithms are the most demanding in terms of processing or communication and how exactly they can be mapped on modern parallel architectures. In particular, although the algorithmic complexity of different sections can be calculated, little has been done in terms of actual performance analysis on real systems. The lack of such knowledge inhibits further progress in this area, since it is not clear how these algorithms should evolve to lead to applicable solutions in the area of ultrasound medical imaging.

In this work we address both of these issues by designing an efficient parallel beamforming algorithm and studying its behavior and requirements on a generic computing architecture that consists of commodity components. First, we review the signal processing algorithm used in the implementation of a 3D ultrasound medical imaging system we are currently building. We provide an efficient, all-software, sequential implementation that shows considerable advantages over hardware-based implementation of the past. We then provide an efficient parallel implementation of the algorithm for a cluster of high-end PCs connected with a low-latency, high-bandwidth interconnection network and study its behavior. The emphasis is on the computational characteristics of the algorithm and the identification of parameters that critically affect both the performance and cost of our system. We study the behavior and performance of the algorithm for a wide set of parameters and we reveal a number of interesting characteristics leading to conclusions about the prospect of using commodity architectures for performing all related processing in this family of medical applications.

Our high level conclusions and contributions are: (i) A 16-processor system today can achieve close-to-real-time performance for high image quality and is certainly expected to do so in the near future. (ii) Only small parts of this family of signal processing algorithms are very computationally intensive. In particular, 85-98% of the time is spent in FFT and beam steering functions for all our runs and most of the runs spent between 92-95% in these functions. (iii) The communication requirements in the particular implementation are fairly small, localized, and certainly within the capabilities of modern low-latency, high-bandwidth interconnects. (iv) Our results provide an indication of the amount of processing required for a given level of image quality and can be used as a reference in designing computing architectures for ultrasound systems.

The rest of the paper is organized as follows: Section 2 provides a background for ultrasound systems. Section 3 presents a comprehensive summary of the family of conventional beamforming algorithms we use. Section 4 describes the platform we use for our experiments. Section 5 describes our sequential and parallel implementations of the algorithm. Section 6 describes our methodology and Section 7 presents our experimental results. Finally we present related work in Section 8 and draw conclusions in Section 9.

2. ULTRASOUND SYSTEMS

Ultrasound medical imaging is one of the most widely used imaging modalities in the area of health services. Ultrasound systems can be used for early diagnosis, screening, monitoring, and minimally-invasive follow up procedures. The ultrasound image quality has dramatically improved the last few years mostly due to the complete elimination of the analogue electronics and the introduction of digital beamforming techniques [24].

Although there is a large base of installed systems and numerous hardware platforms already in use, the majority of these systems share common characteristics. In the near future, practically all ultrasound systems will utilize signal processing techniques to process signals received as a result of the stimulation of the tissue. Such ultrasound systems follow the general structure shown in Figure 1. The system's quality is determined by both the physical characteristics of the system as well as by the signal processing algorithm used to process the signals.

The transducers used in such ultrasound systems are based on phased array transceiver technology. They consist of an array of transceivers which can be aligned in a special geometric configuration such as linear, circular, planar, cylindrical, or spherical. The purpose of the phased array transducer is to exploit the superposition of waves radiated by the individual transceiver of the array's transducer. The ability to control the phase and the amplitude of the ultrasound waves emitted by each individual transceiver allows the angular steering of the radiated beams that are used to illuminate a volume of interest.

After transmitting the sound waves, the ultrasound system comes to the receiving mode. As the sound waves penetrate the volume and encounter objects, reflection occurs. The reflected waves as well as their multi-path versions are received and digitized by the ultrasound machine. A certain segment of the digitized signals is processed by the beamformer, resulting in discrete time series of a certain length. A digital beamformer is a spatial filter that processes data from the array of sensors in order to enhance the signal received from a certain direction reducing the interference of the background noise. Beams are then combined together to form the spatial images of 2D or 3D volumes. Next, we describe in more detail the specific beamforming algorithm we use in our work.

3. BEAMFORMING ALGORITHM

The beamforming algorithm we use in our work is based on the conventional beamforming algorithm [24].

3D planar-phased-array beamformers use multiple beams to scan a 3D volume. The volume is reconstructed by using the transceivers' outputs of the planar array transducer. Each beam is characterized by its angular direction in the

scanned volume. Figure 2 shows how each beam is specified. The thick arrow depicts a beam in the direction (θ, ϕ) in the spherical coordinate system. The center of the coordinate system is the center of the $(N \times M)$ planar array that lies symmetrically on the (X, Y) plane, where N and M denote the number of sensors in each row and column of the array, respectively. The rows and columns of the array are aligned in parallel with the X, Y axis. With each pair of angles (θ, ϕ) we associate another pair of angles (A, B) that are used in the algorithm to characterize a beam. A is the angle between the beam and the X axis and B is the angle between the beam and the Y axis. The boundaries of the volume reconstructed by each beam are specified in terms of these angles A and B (Figure 2). For example, a reconstructed volume is specified to be within $70^\circ \leq A, B \leq 110^\circ$. The number of beams is specified as $a \times b$, where a and b are the numbers of beams in A and B angular directions. The beam width is defined to be the angular width that is covered by a single beam and characterizes the image resolution capabilities of the image reconstruction process. The more beams and narrow beam width the beamformer uses the better quality images it can generate, however, at higher computational costs.

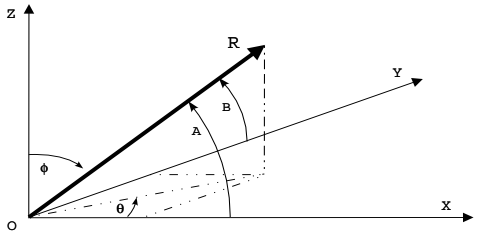

Figure 2: The representation of a beam in the spherical coordinate system.

To produce sharp images of the input object, the scanned volume is divided in multiple focal zones. Ultrasound systems use various focal depths for the reconstructed volume. Each focal zone Z_R is centered around a focal depth R. For example, the zone of depth between 1 cm and 2 cm is reconstructed using a focal depth $R = 1.5$ cm. To produce an image that is focused over the whole reconstructed region, the beamforming process is repeated for each focal zone (characterized by a different value of R). Narrower focal zones produce sharper images but result in more processing as well. In most practical applications, the focal zone size should be in the range between 0.5 and 1.0 cm. In our algorithm we use uniform spacing for R. The beam time series at the output of the beamformer for a specific R is truncated and only the segment that covers the focal zone Z_R under consideration is processed. The images of the different focal zones' Z_R are then concatenated to form the whole volume.

The volume is reconstructed from the time samples of each beam and focal zone as follows. Since the received acoustic wave is coming from a point source located at a finite distance from the array, the wave front is radial. Therefore, the arriving waves are not simple plane waves, but rather spherical waves as being reflected by an object and due to the separation of the transceivers in the array, they arrive at different transceivers with slight time delays. If we assume that the distance of the m^{th} transceiver on the X axis from origin point O is x_m and the distance of the n^{th} transceiver on the Y axis from the same origin point is y_n, then we can compute the time delay between these two transceivers as:

$$t_x = \frac{\sqrt{R^2 + x_m^2 - 2Rx_m \cos A} - R}{c} \tag{1}$$

and

$$t_y = \frac{\sqrt{R^2 + y_n^2 - 2Ry_n \cos B} - R}{c} \tag{2}$$

where c is the speed of sound in human tissue. Thus, for a beam with focal depth of R, the 3D angular response of a N by M planar array to a steered direction (A_x, B_y) can be expressed as:

$$B(f_i, A_x, B_y, R) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I_{m,n}(f_i) S_{m,n}(f_i, A_x, B_y, R) \tag{3}$$

where $I_{m,n}(f_i)$ is the Fourier transform of the input time series from the (m,n) transceiver:

$$I_{m,n}(f_i) = FFT(i_{m,n}(t_i)) \tag{4}$$

and $S_{m,n}(f_i, A_x, B_y, R)$ is the steering vector applied to compensate for the time delay of the (m,n) transceiver with respect to the reference point (located at the center of the planar array):

$$S_{m,n}(f_i, A_x, B_y, R) = e^{j2\pi f_i \left(\frac{\sqrt{R^2 + x_m^2 - 2Rx_m \cos A_x} - R}{c} + \frac{\sqrt{R^2 + y_n^2 - 2Ry_n \cos B_y} - R}{c} \right)} \tag{5}$$

The equation for the angular response (3) can be simplified by separating the term in the steering vector $S_{m,n}$ as follows

$$B(f_i, A_x, B_y, R) = \sum_{n=0}^{N-1} S_n(f_i, B_y, R) \Big[\sum_{m=0}^{M-1} I_{m,n}(f_i) S_m(f_i, A_x, R) \Big] \tag{6}$$

where

$$S_m(f_i, A_x, R) = e^{j2\pi f_i \left(\frac{\sqrt{R^2 + x_m^2 - 2Rx_m \cos A_x} - R}{c} \right)} \tag{7}$$

$$S_n(f_i, B_y, R) = e^{j2\pi f_i \left(\frac{\sqrt{R^2 + y_n^2 - 2Ry_n \cos B_y} - R}{c} \right)} \tag{8}$$

The summation term in square brackets in equation (6) is equivalent to the response of a line array beamformer along the X axis. If we let all the steered beams from this summation term form a vector denoted by $B_n(f_i, A_x)$, then equation (6) can be rewritten as:

$$B(f_i, A_x, B_y, R) = \sum_{n=0}^{N-1} B_n(f_i, A_x) S_n(f_i, B_y, R), \tag{9}$$

which expresses a linear beamforming along the Y axis with $B_n(f_i, A_x)$ as input.

Equation (9) suggests that the 2D planar array beamformer can be decomposed into two linear array beamforming steps. The first step includes a line array beamforming along the X axis and will be repeated N time to get the vector $B_n(f_i, A_x)$. The second step consists of line array beamforming along the Y axis and will be done only once by treating the vector $B_n(f_i, A_x)$ as the input signal for the line array beamformer to get the output $B(f_i, A_x, B_y, R)$.

The decomposition of the planar array beamformer into these two line array beamforming steps leads to an efficient implementation based on the following two factors [24]: First, the number of the involved transceivers for each of these line array beamformers is much smaller than the total number of transceivers, $M \times N$, in the planar array. This kind of decomposition process for the 3D beamformer reduces both memory and CPU requirements. Second, all line array beamformers can be executed in parallel resulting in high degree of coarse-grain parallelism.

Finally, we should note that the number of sensors used in a transducer array is an important parameter for an ultrasound system. Detection of an acoustic signal in a noise field is characterized by the array gain (AG) parameter that is usually defined as:

$$AG = 10\log(M \times N \times BIN)^2$$

where $M \times N$ is the number of sensors and BIN is the number of frequency bins used in the beamforming (or the FFT size as explained later). The more sensors used in a sensor array, the higher is the array gain. The array gain indicates the strength of a beamformer in detecting reflected ultrasound signals. When an object is viewed as a collection of reflective point sources, a beamformer with higher array gain can produce sharper images for the individual point sources.

4. EXPERIMENTAL PLATFORM

Our final ultrasound system follows the overall structure shown in Figure 1. The computing architecture we will use is a modern cluster of high-end PCs. Each node will be equipped with a PCI data acquisition card that will connect the node to a subset of the sensor array. The data acquisition cards will deliver the probing data from the sensor array to the corresponding node's memory. The beamforming algorithm will then reconstruct the image of the scanned object, redistributing data as appropriate. The purpose of this work is to examine the processing component of the system, after the sampled data have been placed in the main memory of each node.

The experimental system we use for evaluation is a cluster of 16 2-way Pentium III nodes interconnected with a Myrinet network [3]. The exact system configuration is summarized in Table 1. Myrinet is a low-latency, high-bandwidth, point-to-point system area network (SAN), used widely for clusters of workstations and PCs. By allowing users to directly access the network, without operating system intervention, Myrinet and other SANs dramatically reduce latencies compared to traditional TCP/IP based local area networks. Moreover, to further reduce latencies in SANs, direct memory operations are usually supported; reads and writes to remote memory are performed without remote processor intervention. Each network interface in our system has a 133 MHz programmable processor (LANai9) and connects the node to the network with two unidirectional links of 160 MByte/s peak bandwidth each. Actual node-to-network bandwidth is usually constrained by the 133 MBytes/s I/O bus on which the NIC sits. All system nodes are connected with a 16-port full crossbar Myrinet switch.

Processors	2 x Intel Pentium III, 800 MHz
Cache	32K (L1), 512K (L2)
Memory	512MB SDRAM
OS	RedHat Linux Kernel 2.2.16-3smp
PCI buses	32 bits, 33 MHz
NIC	Myricom M3M-PCI64B
Communication library	MPICH/Score 4.0

Table 1: Cluster node configuration.

The communication layer we use is the Message Passing Interface (MPI) on top of the SCore system [14]. SCore is a high-performance parallel programming environment for workstation and PC clusters. SCore relies on the PMv2 [26] low-level communication layer. The MPI implementation we use is a port of the MPICH library [19] for the SCore system. Figure 3 shows the bandwidth and latency of the basic, un-contended MPI_Send and MPI_Recv operations. We obtain these point-to-point numbers from running the SKaMPI (Special Karlsruher MPI-Benchmark) benchmark on our system [23]. In all our experiments we use the `gcc` compiler, version `2.91.66`, with the `-O2` optimization level.

5. ALGORITHM IMPLEMENTATION

Our implementations of the algorithm outlined in Section 3 assume that sampled data has already been placed in the main memory of each node by the acquisition units. Next, we present our, in-house, sequential and parallel implementations of the 3D beamforming algorithm.

```
void main()
{
  create_Filter(bf_filter);
  create_SteeringVector(STV);

  //for each tile (pr,pc)
  for(int pr = 0; pr < ROW; pr++) {
   for(int pc = 0; pc < COL; pc++) {
      read_Data(buffer_in[CHC][CHR][NUM_FREQ]);
      while (zone < NZONES) {
        FFT(buffer_in, fft_out);
        Filter(fft_out, bf_filter);
        while(fft samples >= zones samples) {
          for(xb = 0; xb < xBEAMS; xb++) {
            C_Steer(fft_out, STV, az_out);
            for(yb = 0; yb < yBEAMS; yb++) {
              R_Steer(az_out, STV, bout);
              IFFT(bout);
              Write_to(buffer_out, bout);
            }
          }
        }
      }
      display(buffer_out);
    }
  }
}
```

Algorithm 1: Pseudo-code for the sequential implementation.

5.1 Sequential Implementation

Our sequential algorithm for performing the computation

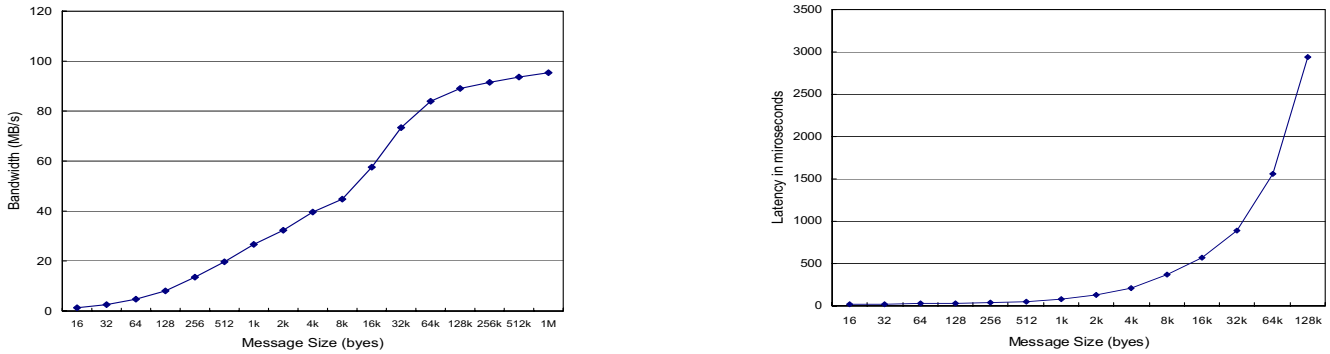

Figure 3: Ping-pong bandwidth (left) and one-way latency (right) for a pair of nodes using (`MPI_Send, MPI_Recv`) to send and receive data.

outlined in Section 3 consists of the following phases: read input samples, compute FFTs, filter results, perform column steering, reorganize data in memory, perform row steering, perform inverse FFTs, and, finally, output to display. Algorithm 1 shows the pseudo-code for this implementation.

In addition to dividing the scanned volume to multiple focal zones and using multiple beams to scan it, beamforming algorithms divide the 2D surface to be scanned in multiple tiles. If we view the 2D surface as an array of points, we can divide the rows and columns in `ROW, COL` groups forming `ROW` $\times$ `COL` tiles. Each tile is scanned by the full planar sensor array. Thus, `CHC` and `CHR` represent the number of sensors in each dimension of the planar array. Every full snapshot (that generates a single full 3D frame of the scanned volume) requires scanning all tiles and focal zones. For real-time processing we would require at least 10 frames (full snapshots) per second and ideally 20 to 30.

To re-create the depth information the algorithm processes the data based on a number of focal zones (`NZONES`). Each focal zone is of constant width (depth), which depends on the depth of the volume to be scanned and the number of focal zones. The volume depth is usually constant and defined by the type of objects the ultrasound will scan. For instance, different human organs require different scan depths. In this work we use a fixed maximum focal depth of 16cm and we vary only the number of focal zones.

For each scanned point of the input volume the program reads the time series data from the corresponding sensor (that may scan multiple points) and stores it in a buffer `buffer_in[CHC][CHR][NUM_FREQ]` in host memory. Each sample is 32 bits and is represented as a single precision floating point number.

The number of samples that need to be processed is dictated by the depth of each focal zone. The probing signal used to scan the object reaches different depths of the scanned volume with different delays. The sampling rate used to digitize the received signal dictates the minimum number of samples (and the minimum FFT size) required to reconstruct depth information. For example, assuming a sampling frequency of $F_s = 30$ MHz, the number of samples needed to reconstruct 20 cm depth can be computed by

$$N = \frac{2dF_s}{c} = \frac{2 \times 0.2 \times 30 \times 10^6}{1540} = 7792,$$

where N is the number of read in samples, c is the speed of sound in meters per second, d is the depth of the reconstruct area in meters. The factor of two is need to account for the round trip time. Thus, in this example the ultrasound system (acquisition unit) needs to provide the beamformer with 7792 samples for each of the sensor time series of the ultrasound probe. Using more than the minimum number of samples can improve the array gain and result in better quality imaging. However, reading in more samples results not only in more processing but also in longer acquisition times and higher storage requirements. In our experiments we set the number of samples to 8K and instead we vary the size of the FFT operations.

After the time samples are read and converted to frequency domain, a filtering phase is used to reduce the amount of information passed to later stages. The information embedded in the received signals necessary for reconstructing a particular depth region is localized in a certain bandwidth. Using only the relevant frequency components further reduces computational time. Thus, the FFT output samples are filtered (with a Finite Impulse Response (FIR) filter [22]) to exclude unnecessary information and the related processing. The bandwidth depends on the focal depth and the center frequency of the ultrasound pulses. Lower frequency signals usually have better penetration into deeper regions, whereas, higher frequency signals produce sharper beam resolutions. However, center frequencies are usually fixed for each depth. Thus, in this work we use 2 MHz as the center frequency (for an input volume with maximum depth of 16 cm). In the applications we are interested in, most objects (human organs) would fall within this range. Given this center frequency the bandwidth of the filter can vary in the range $0.5 - 4.0$ MHz.

After filtering, the beamformer performs the steering operations and finally samples are converted back to the time domain for displaying. Based on equation (6), the `xb` and `yb` loops process the steering of beams on the X and Y axis separately using the pre-calculated steering vector `STV` to align the time delay of the signals arriving in different sensors and IFFT transforms the signal from the frequency to the time domain.

5.2 Optimizations

To gain confidence that we start from an efficient sequential implementation, before proceeding with parallelization, we perform a number of measurements to fine tune several aspects of our sequential implementation.

First, we explore various FFT implementations, both our own and publicly available. We find that, for the processors we use, the most efficient implementation is FFTW [8], a C

```
void main(int argc, char** argv) {
  create_filter(bf_filter);
  create_steering_vector(STV);

  for (each frame tile)
    read_data(buffer_in);
    for (each fft-size samples)
      FFT(buffer_in, fft_out);
      Filter(fft_out, bf_filter);
      for (each focal zone)
        for (each processor proc < NPROCS )
          for (each x-axis beam < xBEAMS/NPROCS)
            C_Steer(fft_out, STV, az_out_send);
          }
          // redistribute data among nodes
          if(proc != My_rank) {
            MPI_Irecv(az_out_buf[proc], sendsize,
                MPI_FLOAT, MPI_ANY_SOURCE, Tag,
                MPI_COMM_WORLD, &recv_req[comm_count]);
            MPI_Send(az_out_send, sendsize, MPI_FLOAT,
                proc, Tag, MPI_COMM_WORLD);
          }
          else {
            memcpy(az_out_buf[proc], az_out_send,
                sendsize*sizeof(float));
          }
        }
        MPI_Waitall(NPROCS-1, recv_req, recv_stat);
        dataReorderTransformation(az_out_buf, az_out);
        for (each x-axis beam < xBEAMS/NPROCS)
          for (each y-axis beam < yBEAMS)
            R_Steer(az_out, STV, bout);
            IFFT(bout);
            write_time_serial_data(buf_out, bout);
          }
        }
      }
      create_display_data(buf_out);
    }
  }
}
```

Algorithm 2: Pseudo-code for the parallel implementation.

library for computing discrete Fourier transforms. Since the input time series data are real numbers we use the real one-dimensional FFT function `rfftw()`. This also minimizes space requirements since the output of this function is a half-complex array that consists of only half the DFT amplitudes; The negative-frequency amplitudes for real data are the complex conjugates of the positive-frequency amplitudes. The side effect of this is that we need to reorganize the output to a common, full-complex array format after the FFT and revert to the half-complex array before the IFFT. Also, FFTW computes an un-normalized transform for the input signal ($IFFT(FFT(x)) = N \times x$) for size N transforms. Thus, a division by N is needed for each element of the array after the final IFFT. The *plan* argument to `rfftw()` is constant across invocations and can be precomputed.

Second, we experiment with multiple ways of performing the steering and the related dot product operations. We notice that the inner summation in equation (6) is actually the summation of M and N complex numbers which are the results of the complex multiplications $I_{m,n} \times (f_i)$ and $S_m(f_i, A_x, R)$, respectively. We find that the best results are obtained by using the `cblas_cdotu_sub`() function from the Intel Math Kernel Library (MKL) [12] to compute the necessary dot products and to perform the steering. MKL is optimized for the Pentium family of processors and makes effective use of the Matrix Manipulation Extensions (MMX) [21], SSE (Streaming SIMD Extensions) [13], and similar instructions.

Third, we tune loop ordering and the layout of multidimensional array data structures to improve memory access behavior and to reduce cache misses.

The overall effect of these optimization steps is a reduction of the overall execution time of the sequential implementation by a factor of about 10. It is a surprising result that hand-tuning can be so effective with all compiler optimizations turned on. However, since in this work we are more interested in the behavior of the parallel version we omit these results due to space limitations.

5.3 Parallel Implementation

The parallel version of the beamforming algorithm follows closely the structure of the sequential implementation. We see that the data read from each sensor is processed independently until steering. Then, during the steering phase, the beams across the X and Y directions are independent. Therefore, we choose to divide the computation in two phases. The first phase includes all processing until after the column-steering phase. The second phase includes the rest of the processing, starting at the row-steering phase. Between the two phases, we need to reorganize the data in memory by performing a matrix transpose, which results in an all-to-all communication pattern.

The first phase of the computation for each frame is decomposed in tasks based on the data generated by each sensor. Thus, there is as many tasks as sensors (e.g. 32×32), which is sufficient for systems with large numbers of processors. The tasks for the second phase are determined by the processing associated with each beam. We use the processing related to a single beam as the basic task and we decompose the second phase to `xBEAMS` $\times$ `yBEAMS` tasks. For instance, with 8 beams in each direction, there are 64 coarse grain tasks. We expect that for all practical applications, at least 8×8 beams will be necessary and thus we do not consider cases where the number of processors is larger than the total number of beams.

We experiment with two implementations of the parallel algorithm. The first implementation uses dedicated nodes for each phase. However, we find that balancing the number of nodes between the two phases of the computation depends on a large number of parameters. Thus, we provide a second, symmetric, implementation as well, where all nodes in the system perform the same type of processing. Although, the first, dataflow approach has certain advantages in reducing task management costs, we find that the second, SPMD approach is more flexible and results in better load balancing. Thus, for the rest of this work we only use our symmetric implementation, as shown in Algorithm 2.

6. EVALUATION METHODOLOGY

The goal of our work is twofold. We are interested in evaluating the absolute performance of this family of algorithms on clusters of generic, commodity components. In addition, we aim to understand the computational characteristics of this emerging class of applications.

Since the data acquisition unit of our system is not available yet and there are no publicly available data from actual systems (due to privacy and other constraints), we use the

Field II ultrasound simulator [15] to generate the input samples for our experiments. The input to the Field II simulator is a point-model of a shell object. For our experiments we use a shell of 50,652 points. The exact simulator parameters for generating the input time-domain signals are shown in Table 2.

Transmit	
Center Frequency	2.0MHz
Bandwidth	2.0MHz
Array Size	12×12
Detector Size	0.35mm
Detector Spacing	0.40mm
Transmit Focal Depth	70mm
Receive	
Array Size	32×32
Detector Size	0.35mm
Detector Spacing	0.40mm
Receive Focal Depth	Infinite (10^{22} m)
Sampling Frequency	33MHz
Shell	
Inner Shell Radius	10mm
Outer Shell Radius	14mm
Shell Center	(5mm, -5mm, 70mm)
Shell Thickness	4mm
Points Defining Shell	50,652
Scatter Density	6.93 pts/mm^3

Table 2: Input parameters for the Field II simulator.

To investigate how each system parameter affects the execution time of the algorithm in practice, we examine the most important system parameters for beamforming-based ultrasound systems. Table 3 summarizes these parameters, their allowable value ranges, and the values used in our experiments. Each parameter is attributed either to the ultrasound system itself (physical) or the beamforming algorithm (algorithmic).

First, we verify that parameters are (for all practical purposes) independent of each other by performing guided experiments (which we do not present here due to space limitations). Thus, we vary each parameter individually by keeping all other parameters constant. The base value and the range we use for each parameter is shown in Table 3. To make the effect of varying each parameter as visible as possible we use as the base case, values that result in relatively low amounts of computation. We denote each configuration with the notation a{$sensors$}-b{$beams$}-f{FFT}-sp{$focal$} -bw{$bandwidth$}, where $sensors$ is the number of sensors in each dimension of the planar array, $beams$ is the number of beams in each tile of a snapshot, FFT is the FFT size, $focal$ represents the focal zone size in millimeters, and $bandwidth$ is the bandwidth of the filter in MHz. For example, the base configuration, denoted as *a32-b8-f512-sp10-bw2.0* specifies a configuration of 32×32 sensor array, 8×8 beams per tile, 512 FFT size, 10 mm depth for each focal zone, and 2.0 MHz filter bandwidth.

It is important to note that changing each parameter impacts not only execution time, but image quality as well. Thus, it is important to be able to quantify image quality and to also take it into account when evaluating various configurations. One traditional method of quantifying image quality is to correlate each generated image with the prototype that is being scanned, and to use the correlation number for ranking output images. In our case, however, since the input time series is generated with the Fields II simulator, there is no actual input object or image. For this reason, we use as the prototype image the best possible image that the algorithm can generate (*a32-b16-f4096-sp05-bw4.0*). We correlate each pair of images by using the same coefficient that is used in statics to express the degree of dependence between two variables [20]. In our case, each variable corresponds to the pixel value of each image. Although it is somewhat simplistic to compare two clinical images just by using the correlation coefficient (without expert opinion from medical personnel), we still get a good indication of the relative quality of various images. We perform various consistency checks to verify that the correlation coefficients correspond, to the extend possible, to the perceived quality of each image and we feel reasonably confident that our ranking methodology is valid.

In our measurements we exclude the initialization time and we present measurements only for the parallel section. Moreover, as mentioned earlier, we assume that input data is delivered to memory by the data acquisition cards. Although these transfers may interfere with other communication in the system, it is not an issue since overall traffic is low (as we will see in Section 7). It is important to note that, although we do not evaluate this aspect of our system, one of the advantages of using a cluster to process the input data, is that the I/O path bandwidth scales linearly as we increase the number of nodes in the cluster. Finally, in our measurements we exclude the time needed to send the processed data from each node to a separate node that displays the images. However, the amount of communication required is very small and occurs over a separate 100 MBit Ethernet network.

For the parallel section of the algorithm, we present both overall execution times as well as execution time breakdowns. To reveal which parts of the algorithm incur high overheads we break execution time to the following components: *FFT* is the total time spent performing FFTs on the input samples. *Filter* is the time filtering frequencies that are outside a pre-specified range. *Csteer* is the time spent steering the samples corresponding to the column sensors. *Communication* is the time spent redistributing the data among the system nodes. For the uniprocessor case, communication time represents the time to transpose the array locally. *Rsteer* is the time spent steering the data that correspond to each sensor row. *IFFT* is the time spent performing inverse FFTs. Finally, *Other* represents the time spent in the rest of the parallel section of the algorithm.

7. RESULTS

In this section we first present our overall performance results and then we examine the effect of each individual parameter separately.

7.1 Overall Execution Time

Figure 4 shows the total execution time of the parallel section of each configuration as the number of processors changes. We see that execution time reduces linearly with the number of processors (the x-axis uses a log scale). This is mainly due to the fact that the partitioning of the tasks is well balanced and the fact that the amount of communication between the two phases of the parallel algorithm is relatively small. The message size depends on the number of the processors, the center frequency of the ultrasound

Parameter	Characteristics	Range	Values used
Number of sensors	Physical	$m \times m (m = 32, 24, 16, 8)$	$\mathbf{32 \times 32}$, 24×24, 16×16, 8×8
FFT size (samples)	Algorithmic	256 - # of samples	**512,** 1024, 2048, 4096
Filter bandwidth (MHz)	Algorithmic	[0.5, 4.0]	0.5, 1.0, 1.5, **2.0,** 2.6, 3.0, 4.0
Focal zones size (cm)	Algorithmic	≤ 1.0	0.5, **1.0**
Number of beams per $10^\circ \times 10^\circ$	Algorithmic	$n \times n \quad (n \leq 16)$	16×16, $\mathbf{8 \times 8}$, 4×4

Table 3: Algorithm parameters, valid ranges for each parameter, and the values we examine. Highlighted values indicate the base value for each parameter.

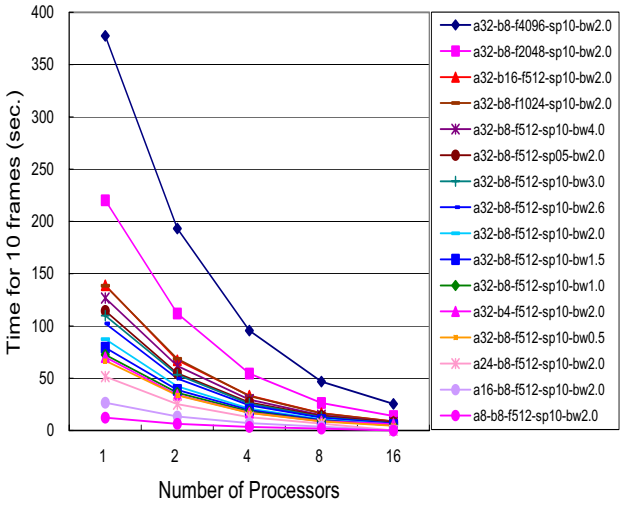

Figure 4: Speedups for different parameter sets by the number of processors.

signal, and the bandwidth of the filter. In our experiments message sizes vary between 112 bytes and 127K bytes. On average, the total amount of data exchanged between the two phases for each frame is about 1 MByte. This imposes fairly small bandwidth requirements on the interconnect and is well within the capabilities of modern system area networks.

Next, we note that the processing rate for our base case, *a32-b8-f512-sp10-bw1.0,* is about 2 frames/s. The configuration with the least amount of processing, *a8-b8-f512-sp10-bw2.0,* can generate about 5.5 frames/s with acceptable quality. Although this is still less than what is needed for real-time performance (ideally, for real-time performance we would require a rate of 20-30 frames/s), using faster processors that are already available would offer 2-3 times better performance today and real-time performance within a few months.

Preliminary runs on a 8-node cluster with 2.0 GHz Pentium4 processors, show that the average speedup compared to our 800 MHz processors varies between 1.6 and 3.9 for an average of about 2.3 across all configurations. For our base configuration, *a32-b8-f512-sp10-bw2.0,* there is a speedup of about 2.1. The speedup on *Csteer* is about 4.0, whereas the speedup on *FFT* and *IFFT* is between 1.5 to 1.7. Finally, our fastest configuration, *a8-b8-f512-sp10-bw2.0,* exhibits a speedup of about 3.2 over our 800 MHz cluster and results in a final speed of about 9 frames/s. Given the linear speedups we observe on our 16-node cluster, we expect that using sixteen 2.0 GHz nodes will result in real-time performance for configuration with acceptable or even high image quality.

7.2 Effects of system parameters

We now describe the effects of different parameters on execution time. Since it is important to also consider the effect on image quality, we also present the correlation rankings for each final (output) image.

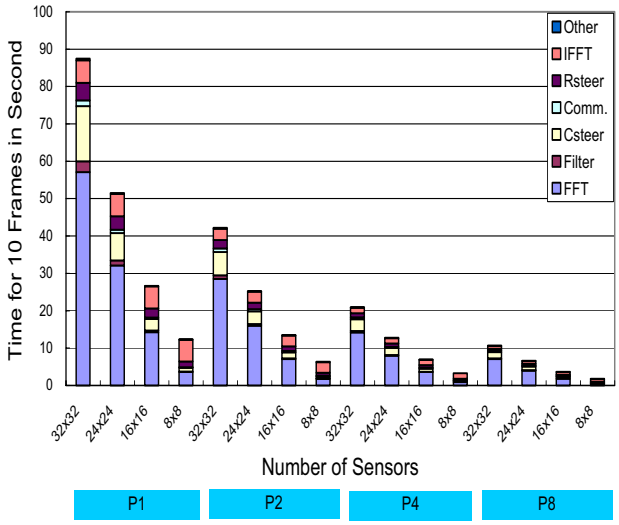

Figure 5: Execution time breakdowns for different numbers of sensors.

7.2.1 *Number of Sensors*

Figure 5 shows the execution time breakdown as we vary the number of sensors. All other parameters are set to their base values, *b8-f512-sp10-bw2.0.* We observe that the overall execution time is almost linear with the total number of sensors and the number of processors. Next, we observe that the time spent in each section of the algorithm reduces linearly with the number of sensors, except for IFFT which is independent of the number of sensors and remains constant.

Figure 6 shows the correlation coefficient for each output image as the number of sensors is reduced (the number of processors does not affect image quality). We see that the image quality degrades significantly as the number of sensors drops; The best and the worst cases differ by more than 15%. However, we should note that whether this reduction in image quality is acceptable for an application, depends on the specific application. For instance, if the objects to be scanned are fairly simple, then the drop in quality may be acceptable, whereas for objects that have more complex contours this may not be the case.

7.2.2 *Number of beams*

Varying the number of beams in the algorithm affects

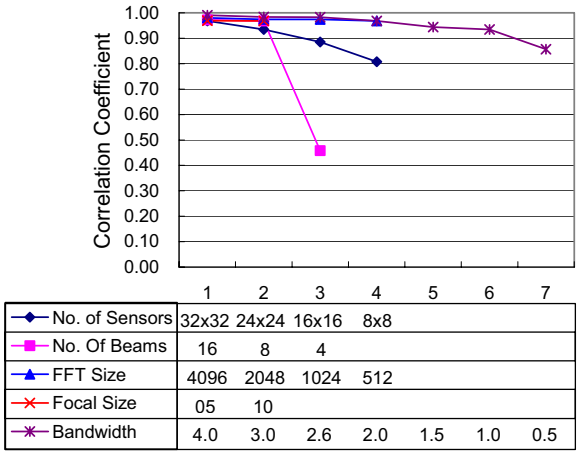

Figure 6: Image correlation coefficient with different parameter sets. The table shows the x-axis values for each curve.

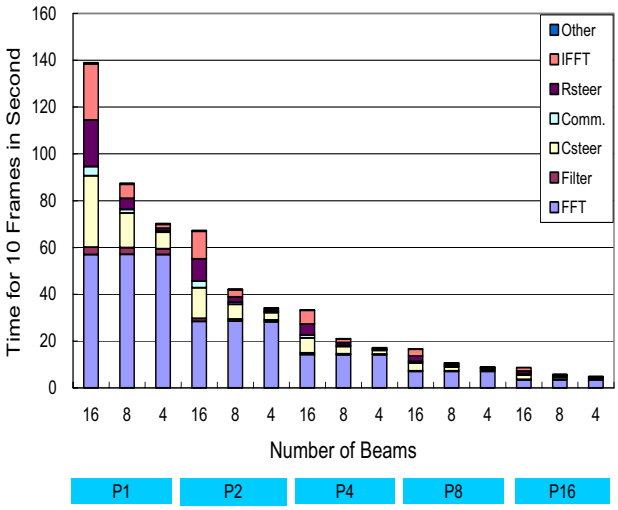

Figure 7: Execution time breakdowns for different numbers of beams.

only the steering operations, the amount of communication, and the inverse FFTs. Figure 7 shows that both row-based steering and IFFT times are reduced super-linearly. This is due to the fact that below a certain number of beams the amount of information to be processed fits completely in the L2 cache. This suggests, that both larger L2 caches can be helpful for this class of applications as well as application knowledge that can be used to limit the number of necessary beams.

The correlation coefficients (Figure 6) show that image quality degrades only if the number of beams is reduced to less than 8. In our experiments, each frame covers an area with an angle of $10^\circ \times 10^\circ$. For the 16×16 beam frames, each beam covers an angle of 0.625°. To cover the same area, the 4×4 beams snapshot has an angle of 2.5° for each beam, which is a lot coarser than using 16×16 beams. Our results suggest that for objects of similar complexity to our input, image quality degrades significantly only if each beam scans more than 1°.

7.2.3 *FFT size*

Figure 8 shows the execution time breakdowns for different FFT size and number of processors. We notice that the overall time spent in FFTs reduces slightly with the FFT size. Although, smaller FFTs result in larger numbers of FFTs and for the sizes we consider the L2 cache is always effective, smaller FFTs tend to be more efficient. We also note that FFT time reduces sub-linearly with the number of processors. Finally, we note that the size of the FFTs affects significantly column and row steering, communication, and inverse FFT times that all reduce linearly with FFT size.

Figure 6 shows that FFT size has practically no influence on image quality for the input we use. The reason for this is that the time samples have a high gain even for small FFT sizes and the algorithm is able to reconstruct a sharp image of the input. We expect that this behavior will change when we use input objects with more complex contours in the actual system. However, these results indicate that understanding the application areas where the ultrasound system is used, can help identify appropriate values for parameters such as the FFT size and to optimize for system cost, performance, and image quality tradeoffs.

7.2.4 *Focal zone size*

Similarly to FFT size, the focal zone size affects only the second phase of the algorithm. Decreasing the focal zone size from 10 to 5 mm doubles the number of focal zones (from 16 to 32) that are required to cover the same depth of volume (16cm) and increases the time required for the second phase of the algorithm linearly (Figure 9). Finally, image quality is not affected significantly by the focal zone size (Figure 6), for reasons similar to what was explained for the effects of the FFT size.

7.2.5 *Filter bandwidth*

Changing the filter bandwidth affects all aspects of the algorithm, except for the time spent in FFTs and IFFTs (Figure 10). The correlation coefficient for the output images (Figure 6) shows that image quality degrades only if the filter bandwidth drops below 1.0 MHz. This is due to the physical characteristics of our simulated transducer array. The acoustic signal we use has a bandwidth of 2.0 MHz which results in useful information being contained in a 2.0 MHz bandwidth in the frequency domain after the Fourier transform of the input samples. Using smaller filter bandwidths excludes some of this information, and bandwidths less than 1.0 MHz result in significant degradation of image quality.

7.2.6 *Summary*

Overall, we find that our parallel implementation scales linearly with the number of processors and that we can achieve almost real-time performance with state-of-the-art clusters. Furthermore, we find that the amount of time spent in FFT and steering operations dominates. In all our experiments, the parallel implementation spends 85-98% of the time in FFT and beam steering functions and most of the runs spent between 92-95% in these functions. Finally, the values of different parameters have significant impact on the computational requirements of the algorithm. Thus, application knowledge that can help selecting appropriate values

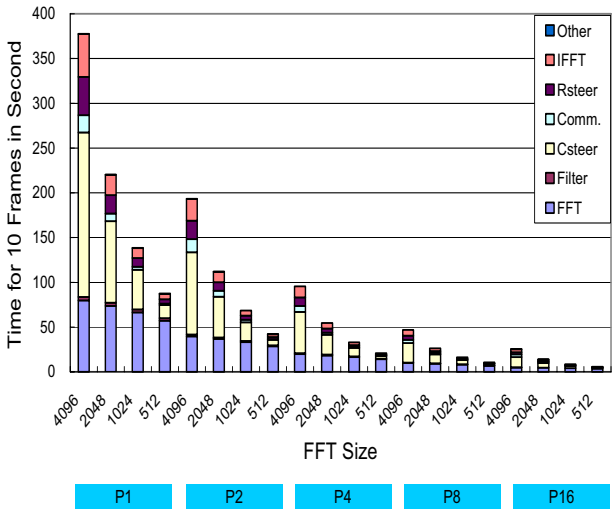

Figure 8: Execution time breakdowns for different FFT sizes.

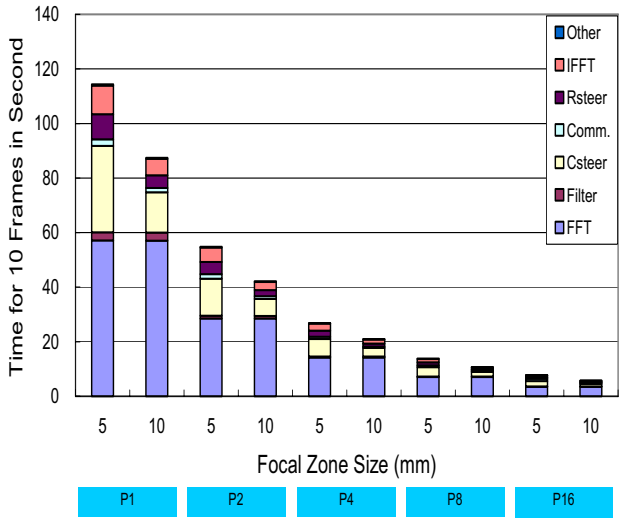

Figure 9: Execution time breakdowns for different focal zone sizes.

for these parameters may be important in optimizing future ultrasound systems for cost and performance.

8. RELATED WORK

To the best of our knowledge, there is very little work in understanding the computational characteristics of ultrasound imaging beamforming processing algorithms on modern clusters. Numerous solutions for the acquisition problem and a large number of algorithms for processing the sensor time series have been proposed recently [25, 11, 2, 15, 18, 24]. This work has examined, among other, issues related to transducers and their relation to beamforming techniques for ultrasound systems. Our work is orthogonal to this and relies on high-quality transducer arrays. Also, previous work has examined the usage of beamforming algorithms in ultrasound and other medical applications [17, 7]. Finally, there has been a large body of work on parallel beamforming algorithms and implementations on both high-end parallel systems and distributed workstations [4, 5, 9, 16, 1]. However, all this work has examined applications from other domains, and in particular sonar systems.

9. CONCLUSIONS

In this paper we examine a family of algorithms that are used in high-resolution 3D medical imaging systems. We present the necessary background, we describe the fundamental algorithmic aspects, and study the computational behavior on modern architectures. Our work, indicates that for many applications, specialized architectures are not necessary and that generic clusters may be used.

In particular, we see that our implementation of a state-of-the-art beamforming algorithm, by carefully decomposing the original problem, results in linear speedups in systems up to 16 processors. On a 16-processor system we can achieve almost-real-time medical imaging with acceptable or high image quality. Given that we use older-generation processors, we expect that today's systems (or within a few months) will be able to provide real-time performance, resulting in significant flexibility and cost benefits compared to traditional custom solutions. Preliminary results with 2.0 GHz Pentium4 processors show that there is an average speedup of about 2.3 across configuration compared to our 800 MHz cluster. This ability to take advantage of the latest system components that become available with no additional costs for re-designing the system architecture is one of the fundamental benefits of our approach in addressing issues in this area. Thus, given our results, we expect that modern clusters will be used with success in a wider range of medical applications.

We find that FFT and steering costs are the most significant overheads and that communication requirements are very low. In most of our experiments, the application spends between 92-95% in these sections. Furthermore, we study and reveal how each section of the parallel implementation depends on system parameters. We find that most dependences are linear with small super- or sub-linear effects. In terms of the induced communication, each processor exchanges a small number of messages with all other processors in the communication phase. We use correlation coefficients to quantify the impact on image quality and we find that the effects of different parameters on image quality is very diverse and indicates that application knowledge is important in optimizing future ultrasound systems for cost-performance. Furthermore, our work indicates that if specialized solutions are necessary, for instance, portable ultrasound systems, system designers can focus on optimizing certain sections of the algorithm and ignoring the rest.

Finally, we expect that, given their advantages over more traditional solutions, modern clusters with low-latency and high-bandwidth networks will be capable of handling a wide range of medical applications and they will be instrumental in improving the cost and precision of medical infrastructure.

10. ACKNOWLEDGMENTS

We are thankful to Andreas Moshovos for helping with various, uniprocessor optimizations of the sequential algorithm. We thankfully acknowledge the support of Natu-

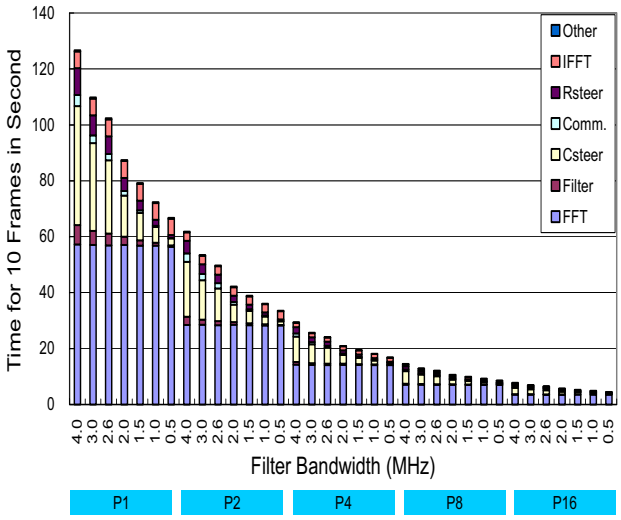

Figure 10: Execution time breakdowns for different filter bandwidths.

ral Sciences and Engineering Research Council of Canada, Canada Foundation for Innovation, Ontario Innovation Trust, the Nortel Institute of Technology, Communications and Information Technology Ontario, and Nortel Networks.

11. REFERENCES

[1] G. E. Allen and B. L. Evans. Real-time sonar beamforming on a unix workstation using process networks and posix threads. *IEEE Transactions on Signal Processing*, March 1999.

[2] B. Angelsen, H. Torp, S. Holm, K. Kristoffersen, and T. Whittingham. Which transducer array is best. *European Journal of Ultrasound*, 2:151–164, 1995.

[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. K. an d C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[4] J. Chapin, A. Chiu, and R. Hu. PC Cluster for Signal Processing: An Early Prototype. *Proceedings of the 2000 IEEE Sensor Array and Multichannel Signal Processing Workshop*, pages 525–529, March 2000.

[5] H. Cox, R. M. Zeskind, and M. M. Owen. Robust adaptive beamforming. *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-35(10):1365, 1987.

[6] D. Dunning and G. Regnier. The Virtual Interface Architecture. *Proceedings of Hot Interconnects V Symposium*, August 1997.

[7] S. Freeman, M. Quick, M. Morin, C. Anderson, C. Desilets, T. Linnenbrink, and M. O'Donnell. Efficient, high-performance ultrasound beamforming using oversampling. *IEEE Proceedings of the 1998 SPIE Medical Imaging Conference*, 1998.

[8] M. Frigo and S. G. Johnson. FFTW: An Adaptive Software Architecture for the FFT. *Proc. of The 2002 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 3:1381–1384, 1998.

[9] A. D. George, J. Markwell, and R. Fogarty. Real-time sonar beamforming on high-performance distributed computers. *Parallel Computing*, 26(10):1231–1252, 2000.

[10] Giganet. Giganet cLAN Family of Products. http://www.emulex.com/products.html, 2001.

[11] S. Holm. Medical ultrasound transducers and beamforming. *Proceedings of The International Congress on Acoustics*, June 1995.

[12] Intel Corporation. Intel Math Kernel Library Reference Manual, 2001. http://developer.intel.com/software/products/mkl.

[13] Intel Corporation. Streaming simd extensions - matrix multiplication. Order Number: 245045-001, June 1999.

[14] Y. Ishikawa, H. Tezuka, A. Hori, S. Sumimoto, T. Takahashi, F. O'Carroll, and H. Harada. RWC PC Cluster II and SCore Cluster System Software – High Performance Linux Cluster. *Proceedings of the 5th Annual Linux Expo*, pages 55–62, 1999.

[15] J. A. Jensen. Field: A Program for Simulating Ultrasound Systems. *Medical and Biological Engineering and Computing*, 34:351–353, 1996.

[16] D. H. Kim. Use of the message passing interface in a real-time sonar beamformer. *DoD High Performance Computing Modernization Program, Users Group Conference*, June 1999.

[17] J. Lu, Z. Hehong, and J. Greenleaf. Biomedical ultrasound beamforming. *Ultrasound in Medicine & Biology*, 20(5):403–428, 1994.

[18] T. Nelson, D. Downey, D. Pretorius, and A. Fenster. *Three Dimensional Ultrasound*. Lippineott Williams & Wilkins, Philadelpia, 1999.

[19] F. O'Carroll, H. Tezukua, A. Hori, and Y. Ishikawa. MPICH-PM: Design and Implementation of Zero Copy MPI for PM. *Technical Report TR-97011, Real World Computing*, March 1998.

[20] A. Papoulis. *Probability, Random, Variables, and Stochastic Processes*. McGraw-Hill, Inc., New York, NY, 1984.

[21] A. Peleg and U. Weiser. Mmx technology extension to the intel architecture. *IEEE Micro*, 16(4):42–50, 1996.

[22] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Macmillan Publishing Company, New York, NY, 1992.

[23] R. H. Reussner, P. Sanders, L. Prechelt, and M. Müller. SKaMPI: A detailed, accurate MPI benchmark. *Recent advances in parallel virtual machine and message passing interface: 5th European PVMMPI Users' Group Meeting*, 1497:52–59, September 1998.

[24] S. Stergiopoulos (Editor). *Advanced Signal Processing Handbook: Theory and Implementation For Radar, Sonar, and Medical Imaging Real Time Systems*. CRC Press, Boca Raton, FL, 2000.

[25] S. Smith, R. Davidsen, C. Emery, R. Goldberg, and E. Light. Update on 2-d array transducers for medical ultrasound. *Proceedings of The IEEE Symposium on Ultrasound*, pages 1273–1278, 1995.

[26] T. Takahashi, S. Sumimoto, A. Hori, H. Harada, and Y. Ishikawa. PM2: High Performance Communication Middleware for Heterogeneous Network Environment. *Proc. of The Supercomputing 2000 Conference (SC2000)*, November 2000.