Chen *et al. EURASIP Journal on Embedded Systems* (2017) 2017:19
DOI 10.1186/s13639-017-0069-7

EURASIP Journal on Embedded Systems

RESEARCH Open Access

Design and evaluation of medical ultrasonic adaptive beamforming algorithm implementation on heterogeneous embedded computing platform

Junying Chen*, Jinhui Chen and Huaqing Min

Abstract

Medical ultrasonic imaging has been utilized in a variety of clinical diagnoses for many years. Recently, because of the needs of portable and mobile medical ultrasonic diagnoses, the development of real-time medical ultrasonic imaging algorithms on embedded computing platforms is a rising research direction. Typically, delay-and-sum beamforming algorithm is implemented on embedded medical ultrasonic scanners. Such algorithm is the easiest to implement at real-time frame rate, but the image quality of this algorithm is not high enough for complicated diagnostic cases. As a result, minimum-variance adaptive beamforming algorithm for medical ultrasonic imaging is considered in this paper, which shows much higher image quality than that of delay-and-sum beamforming algorithm. However, minimum-variance adaptive beamforming algorithm is a complicated algorithm with $O(n^3)$ computational complexity. Consequently, it is not easy to implement such algorithm on embedded computing platform at real-time frame rate. On the other hand, GPU is a well-known parallel computing platform for image processing. Therefore, embedded GPU computing platform is considered as a potential real-time implementation platform of minimum-variance beamforming algorithm in this paper. By applying the described effective implementation strategies, the GPU implementation of minimum-variance beamforming algorithm performed more than 100 times faster than the ARM implementation on the same heterogeneous embedded platform. Furthermore, platform power consumptions, computation energy efficiency, and platform cost efficiency of the experimental heterogeneous embedded platforms were also evaluated, which demonstrated that the investigated heterogeneous embedded computing platforms were suitable for real-time portable or mobile high-quality medical ultrasonic imaging device constructions.

Keywords: Embedded GPU implementation, Medical ultrasonic adaptive beamforming, High-performance computing

1 Introduction

There are several useful medical imaging modalities for clinical diagnoses, which are radiography [1], magnetic resonance imaging [2], nuclear imaging [3], ultrasonic imaging [4], and computed tomography [5]. Among the mentioned medical imaging modalities, medical ultrasonic imaging is a very common imaging technique which has been widely applied to a vast range of clinical diagnoses for many years [6]. Medical ultrasonic imaging has overwhelming advantages over other medical imaging modalities, such as real-time imaging at a smooth video frame rate, high safety without electromagnetic radiation, and relatively low cost as compared to other medical imaging modalities [7]. Therefore, medical ultrasonic imaging is usually utilized to observe heart movements [8], fetal developments [9], blood flows [10], and so on. Recently, as the needs for portable and mobile medical ultrasonic diagnoses increase, the development of medical ultrasonic imaging algorithms on embedded computing platforms is a rising research direction. The key evaluation feature of an embedded medical ultrasonic

*Correspondence: jychense@scut.edu.cn
Guangzhou Key Laboratory of Robotics and Intelligent Software, School of Software Engineering, South China University of Technology, Guangzhou, China

imaging algorithm implementation is its real-time imaging capability. A medical ultrasonic imaging algorithm can be used for real-world clinical diagnostic applications only if its implementation can run at a real-time video frame rate.

Delay-and-sum (DAS) beamforming [6] is the most widespread imaging algorithm among various medical ultrasonic imaging algorithms, which is also the easiest to implement at real-time frame rate. But the image quality of DAS algorithm is not high enough for complicated diagnostic cases. On the other hand, high-quality medical ultrasonic imaging algorithms (e.g., minimum-variance (MV) adaptive beamforming algorithm [11], synthetic aperture imaging algorithm [12]) provide much better image quality, as compared to DAS beamforming algorithm, which can present more anatomical details for complicated diagnostic cases. For example, Fig. 1 illustrates the advantage of MV adaptive beamforming over DAS beamforming. The example used field II simulator [13] to generate the simulated imaging input data. Field II simulator is a widespread tool to generate reliable medical ultrasonic imaging input data, especially for quantitative evaluation of the output image quality and verification of imaging algorithm correctness, which is its major benefit over real scenario. As shown in Fig. 1, DAS beamforming failed to resolve the two very close point targets at 30-mm imaging depth, making the two points look like a short line instead. But on the other hand, the two very close point targets at 30-mm imaging depth were clearly distinguished using MV adaptive beamforming algorithm.

However, these high-quality imaging algorithms are computationally demanding and difficult to implement in real-time, especially on embedded platforms with limited computing resources. The real-time implementation of MV adaptive beamforming is challenging, whose computational complexity is $O(n^3)$ in sequential implementation [14]. Among various embedded computing platforms, field-programmable gate arrays (FPGAs) are good potential development platforms to implement real-time medical ultrasonic imaging algorithms. However, an FPGA platform with sufficient computational resources to implement a regular high-quality medical ultrasonic imaging algorithm at real-time video frame rate is usually of high price. Hence, in order to reduce the implementation cost of a portable or mobile medical ultrasonic imaging device, alternative high-performance embedded computing platform with relatively lower implementation cost is required. As a result, the heterogeneous embedded computing platforms with high-performance graphics processing units (GPUs) and advanced RISC machine (ARM) processors will be investigated in this paper.

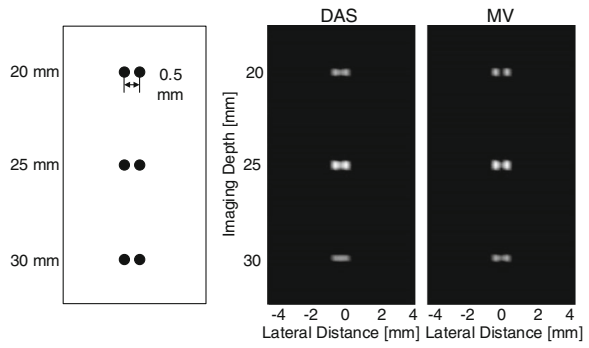

Fig. 1 Results of imaging two very close point targets at different imaging depths using DAS beamforming algorithm and MV adaptive beamforming algorithm

In this paper, the implementation of MV adaptive beamforming algorithm on high-performance embedded GPU computing platform will be discussed. The implementation strategies for a high-performance GPU on a heterogeneous embedded computing platform will be described, and the performance of the GPU implementation and its ARM processor counterpart on the same embedded computing platform will be evaluated. The performance features evaluated in this paper will include the algorithm computing speed, the computation energy efficiency, and the platform cost efficiency. The following section will interpret the detailed sequential calculation steps of the medical ultrasonic MV adaptive beamforming algorithm. Then, the implementation design of the MV adaptive beamforming algorithm on the embedded GPU computing platform will be described in Section 3. Section 4 will illustrate the experimental setup and the performance evaluations of the embedded implementations. Finally, the paper concludes in Section 5.

2 Medical ultrasonic adaptive beamforming sequential calculation

The adaptive beamforming algorithm investigated in this paper is based on the principles of medical ultrasonic MV adaptive beamforming algorithm described in [11, 14]; the MV adaptive beamforming algorithm is implemented according to the calculation steps illustrated in Fig. 2.

The MV adaptive beamforming algorithm starts with the input matrices which contain unaligned ultrasonic echo signals from a series of ultrasonic echo channels. The number of the ultrasonic echo channels is denoted as M, which is also called the receive aperture of the ultrasonic echo signals. In order to obtain the aligned $(M \times 1)$ echo signal vector $\mathbf{echo}(p_s)$ for a specific image pixel p_s, the algorithm calculates the delay information of all the ultrasonic echo channels for the specific image pixel.

In the practical MV adaptive beamforming implementation, sub-aperture averaging is a critical technique which helps to increase the image quality. When sub-aperture averaging is applied, the receive aperture M is segmented into a set of sub-apertures. If the size of the sub-aperture

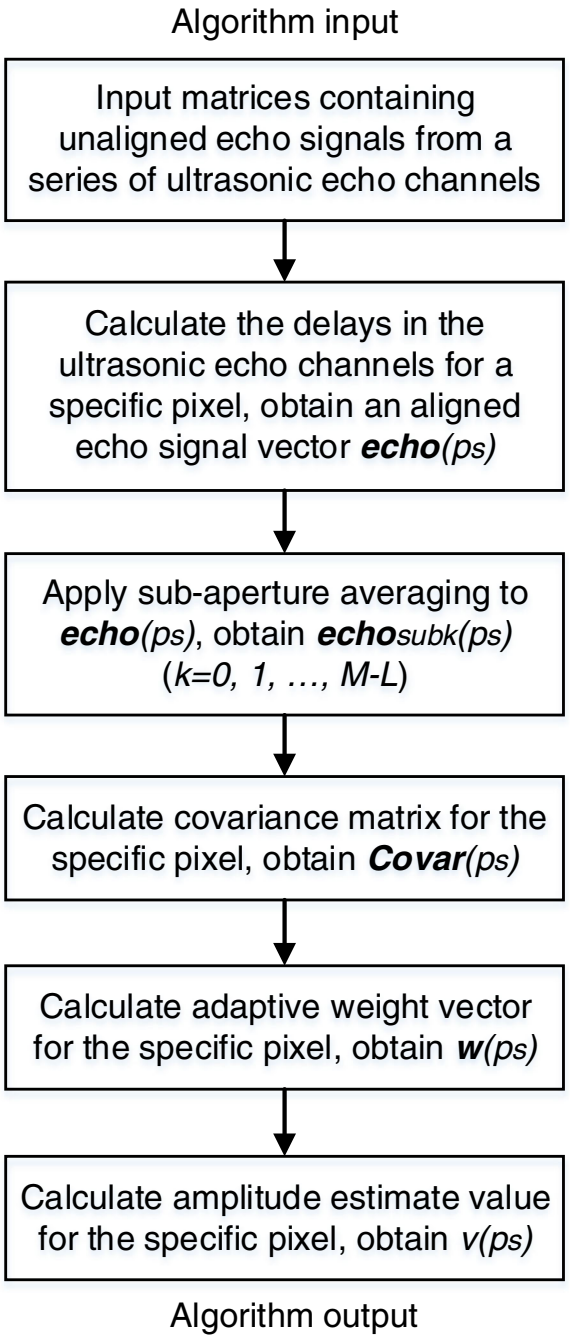

Fig. 2 Medical ultrasonic MV adaptive beamforming algorithm calculation steps (Calculation steps repeat until all pixel amplitude estimate values are obtained)

is L, which means there are L consecutive ultrasonic echo signal channels within the sub-aperture, $(M - L + 1)$ sub-apertures are constructed. As a result, the aligned ultrasonic echo signal vector $\mathbf{echo}(p_s)$ is segmented into $(M - L + 1)$ sub-aperture echo signal vectors $\mathbf{echo}_{\mathrm{subk}}(p_s)(k = 0, 1, \ldots, M - L)$, where $\mathbf{echo}_{\mathrm{subk}}(p_s)$ is a $(L \times 1)$ vector of input ultrasonic echo signals in kth sub-aperture, i.e., $\mathbf{echo}_{\mathrm{subk}}(p_s)$ is the assemble of kth to $(k + L - 1)$th elements of $\mathbf{echo}(p_s)$ vector. Then, the covariance matrix calculation with sub-aperture averaging for a specific image pixel is expressed as

$$\mathbf{Covar}(p_s) = \frac{\sum_{k=0}^{M-L} \mathbf{echo}_{\mathrm{subk}}(p_s)\mathbf{echo}_{\mathrm{subk}}^{H}(p_s)}{M - L + 1}. \quad (1)$$

The calculation of adaptive apodization weight vector is conducted after the covariance matrix $\mathbf{Covar}(p_s)$ is obtained, which is calculated as

$$\boldsymbol{w}(p_s) = \frac{\mathbf{Covar}^{-1}(p_s)\boldsymbol{a}}{\boldsymbol{a}^{H}\mathbf{Covar}^{-1}(p_s)\boldsymbol{a}}, \quad (2)$$

where $\boldsymbol{a}$ is a steering vector with all ones, which is because the ultrasonic echo signals used for covariance matrix calculation are already delayed and aligned.

Finally, when the adaptive apodization weight vector $\boldsymbol{w}(p_s)$ is ready, the amplitude estimate value of the specific image pixel p_s is obtained by

$$v(p_s) = \frac{1}{M - L + 1} \sum_{k=0}^{M-L} \boldsymbol{w}^{H}(p_s)\mathbf{echo}_{\mathrm{subk}}(p_s). \quad (3)$$

The pixel amplitude estimate value $v(p_s)$ is the output of the MV adaptive beamforming algorithm. The calculation steps repeat until all the pixel amplitude estimate values of the whole image are obtained.

3 Implementation design on embedded GPU computing platform

3.1 Heterogeneous embedded computing platform

Although MV adaptive beamforming algorithm outputs high-quality medical ultrasonic images, it is computationally demanding. The high computational complexity of MV adaptive beamforming algorithm hinders MV algorithm being implemented in real-time on conventional embedded computing platforms, such as conventional ARM processors. Therefore, the implementation of MV adaptive beamforming algorithm on heterogeneous embedded computing platform with high-performance GPU is explored in this paper, so as to validate the real-time imaging capability of MV adaptive beamforming algorithm on embedded platforms.

The symbolic architecture modules of the heterogeneous embedded computing platform is illustrated in Fig. 3. As shown in Fig. 3, the ARM processors and the embedded GPU are within one single embedded processing chip, as well as the internal memory and the external memory modules. There are plenty of peripherals on the heterogeneous embedded computing platform, such as camera input module, display output module, USB, GPIO, and other common peripheral connector modules. The heterogeneous embedded computing platforms investigated in the MV adaptive beamforming

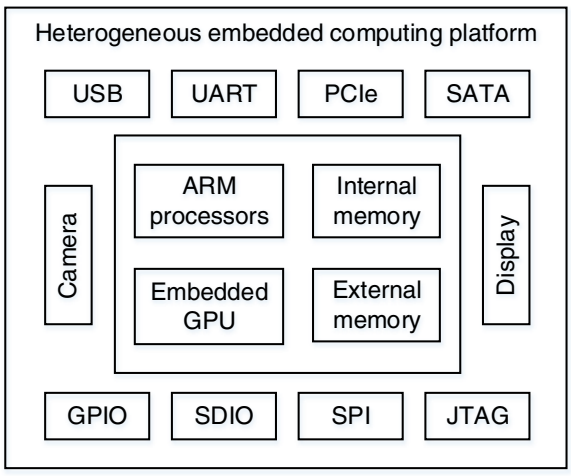

Fig. 3 Heterogeneous embedded computing platform architecture modules

algorithm implementation experiments and evaluations are shown in Section 4, which are the products of Nvidia Corporation.

3.2 Implementation strategies for embedded GPU

Before starting the implementation design of MV adaptive beamforming algorithm on the embedded GPU computing platform, the sequential MV adaptive beamforming code execution time profiling was carried out, so as to determine which parts were the time-consuming parts of the sequential algorithm code. The execution time profiling result is shown in Fig. 4. As seen from Fig. 4, the input data delay calculation and the covariance matrix calculation consumed most of the execution time of the sequential MV code. As L/M increased, the percentage of execution time used for covariance matrix calculation and adaptive weight calculation increased, but the percentage of execution time for input data delay calculation decreased. Such changes conformed to the computational complexities of different calculation steps. For example, the input data delay calculation had a computational complexity of $O(n)$, the covariance matrix calculation had a computational complexity of $O(n^2)$, and the adaptive weight calculation had a computational complexity of $O(n^3)$. Based on the analysis of the sequential MV code, the code parts with high computational complexity or high percentage of execution time should be implemented on the GPU platform. As a result, also considering reducing the hardware/software communications between GPU and ARM processor, all parts of the MV adaptive beamforming algorithm were decided to implement on the GPU platform.

In order to effectively utilize the high-performance embedded GPU in the heterogeneous embedded computing platform to implement the MV adaptive beamforming algorithm efficiently, the following implementation strategies are applied.

3.2.1 *Allocation of GPU computing resources*

The first implementation strategy is about the allocation of the GPU computing resources. According to the GPU compute unified device architecture (CUDA) programming principles, the GPU CUDA programming model consists of three programming hierarchy levels, i.e., GPU compute grid, GPU compute block, and GPU compute thread. At the top level of the programming hierarchy, all the algorithm computations are executed within one GPU compute grid. Meanwhile, at the second level of the programming hierarchy, the program tasks are allocated into a set of GPU compute blocks. The computation in different GPU compute blocks can be executed in parallel. Besides, at the third level of the programming hierarchy, the computational workloads are assigned to a series of GPU compute threads. The programs in different GPU compute threads are executed simultaneously, while the program instructions within one GPU compute thread are executed sequentially.

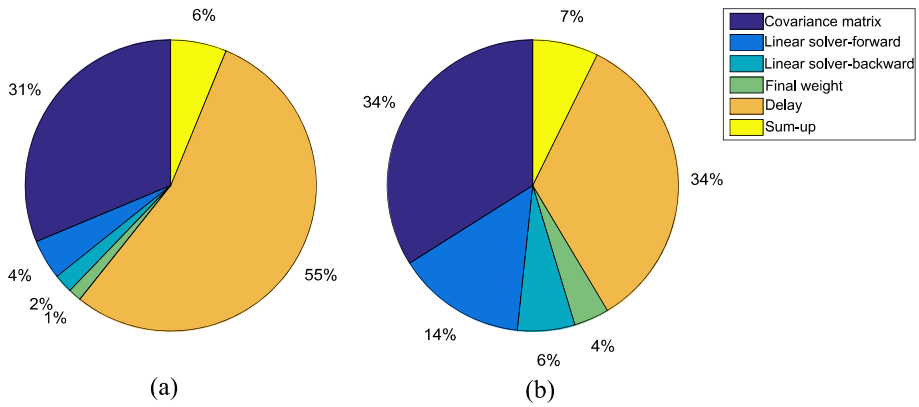

Fig. 4 Sequential MV adaptive beamforming code execution time profiling. **a** $L/M = 1/4$. **b** $L/M = 1/2$

Such hierarchical GPU CUDA programming model can be applied to the medical ultrasonic image formation process which utilizes MV adaptive beamforming algorithm. In the medical ultrasonic image formation process, the image pixel amplitude estimate values of the whole image are calculated following MV adaptive beamforming algorithm calculation steps as described in Section 2. The image pixels are organized in rows and columns, which can be mapped to the GPU compute grid with a two-dimensional GPU compute block allocation, as shown in Fig. 5. As a result, each GPU compute block is responsible to the computation of one image pixel amplitude estimate value calculation. The program steps used to calculate the image pixel amplitude estimate value are executed inside one block via the simultaneous computation of the parallel threads within the block. Finally, the sequential operations of the program, which cannot be executed at the same time, are computed inside the threads. The best practices of the GPU block size and the GPU thread size rely on the arrangement of the computational resources on the embedded computing platform according to the computational problem size.

3.2.2 Overall design overview

Referring to the allocation of GPU computing resources illustrated in Fig. 5, the high-level design block diagram of the MV adaptive beamforming algorithm for GPU implementation is demonstrated in Fig. 6. Based on the overall design block diagram, the fine-grained parallelization of the algorithm implementation was conducted on the embedded GPU.

In the design, the MV beamforming implementation took M receive channels as its inputs to generate an amplitude estimation of one image pixel. The input data from the ultrasonic echo receive channels was also known as pre-beamform data before the beamforming process, and the output pixel value was also known as post-beamform data after the beamforming process.

Each receive channel streamed input echo samples to the delay calculation block, forming an $M \times 1$ vector of delayed echo samples as its output. The purpose of the delay calculation block was to align the input ultrasonic echoes based on the delay information among the receive channels. The delayed echo vector must subsequently be multiplied by the adaptive apodization weights. Hence, while the adaptive weight calculation block was calculating the adaptive weights, the echo vector was stored in the data storage buffer for later usage, which was built with GPU shared memory. The purpose of data storage buffer was to let the delayed echo vector wait until the adaptive weight calculation block finished its workload.

Finally, the pixel amplitude estimation block outputs the final pixel value. It first multiplied $(M - L + 1)$ segmented delayed echo vectors $\mathbf{echo}_{\mathrm{subk}}(p_s)(k = 0, 1, \ldots, M - L)$

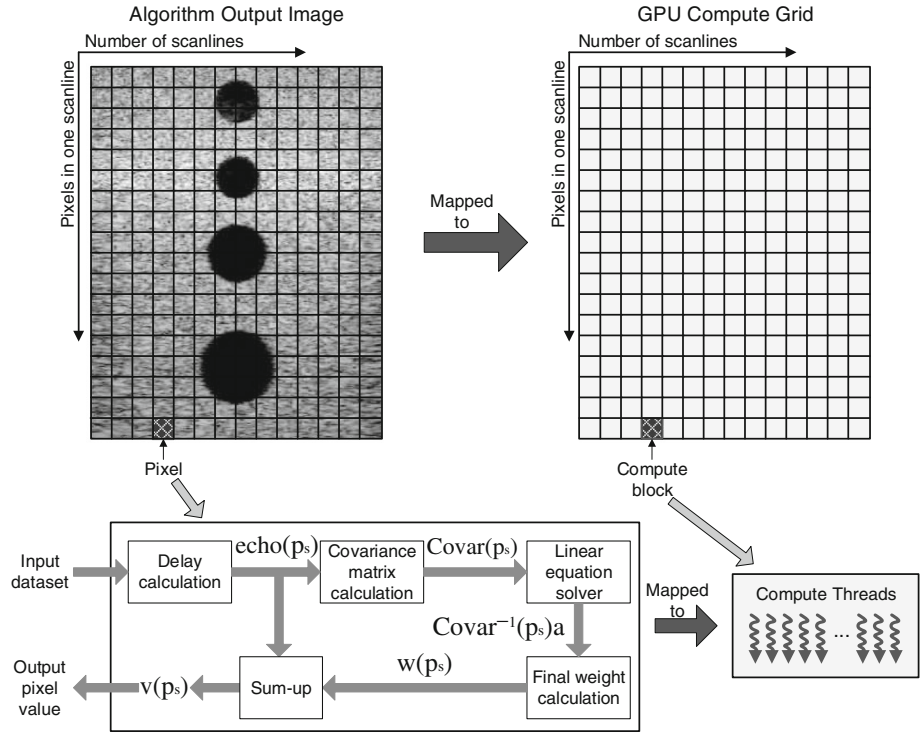

Fig. 5 The imaging pixel mapping to GPU computing resources

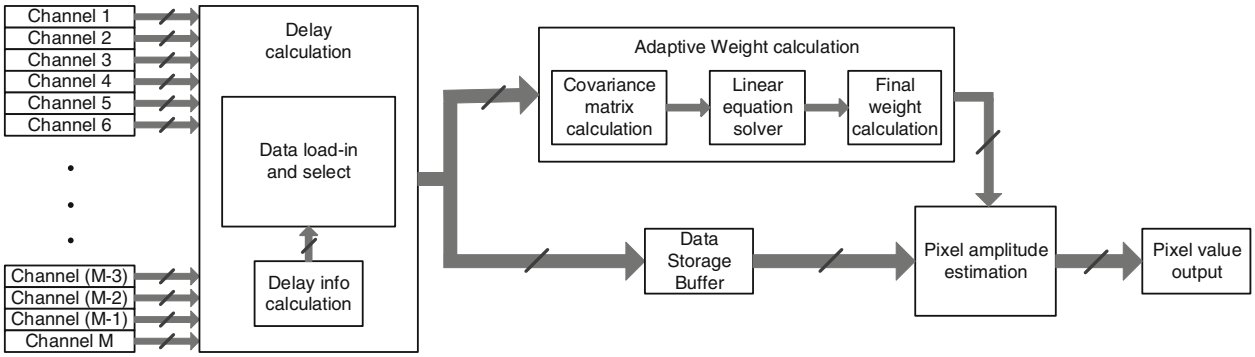

Fig. 6 Design block overview of MV adaptive beamforming algorithm on GPU computing platform

and their adaptive weights. The results of these $(M - L + 1)$ pixel value estimates were then averaged to obtain the final pixel amplitude value output.

3.2.3 Adaptive weight calculation parallelization

In order to implement a highly parallelized MV algorithm, probability theories and linear algebra theories were used to optimize the detailed implementation and reduce the computation operations. The integration of the mathematical theories into the implementation will be described in the following three parts.

The adaptive weight calculation block in Fig. 6 performs the core computation of the MV adaptive beamforming algorithm. It consists of three major units: covariance matrix calculation, linear equation solver, and final weight calculation. Here, the inner working principles of these blocks will be elaborated.

Covariance matrix calculation Derived from (1), the covariance matrix element $\text{Covar}_{ij}(p_s)$ is calculated as

$$\text{Covar}_{ij}(p_s) = \frac{\sum_{k=0}^{M-L} \text{echo}_{(i+k)}(p_s)\text{echo}_{(j+k)}(p_s)}{M - L + 1}, \quad (4)$$

where $\text{echo}_{(i+k)}(p_s)$ is the $(i + k)$th element in $\mathbf{echo}(p_s)$ vector. As the input digital echo data are real numbers, the covariance matrix is a symmetric matrix [15], which has the following property:

$$\mathbf{Covar}^T(p_s) = \mathbf{Covar}(p_s). \quad (5)$$

As a result, only the diagonal elements and the lower (**L**) or upper (**U**) triangular matrix elements of the covariance matrix need to be calculated. Therefore, $L \times (L + 1)/2$ calculations are needed in stead of $L \times L$ calculations. Taking the advantage of the symmetry makes the covariance matrix implementation nearly twice faster.

Linear equation solver As shown in the weight calculation (2), $\mathbf{Covar}^{-1}(p_s)$ has to be calculated. But also shown in (2), every $\mathbf{Covar}^{-1}(p_s)$ is multiplied by a vector $\boldsymbol{a}$. As a result, a linear equation solver which outputs $\mathbf{Covar}^{-1}(p_s)\boldsymbol{a}$ can take over the places of the matrix inverse unit and the matrix multiplication unit. The solver is used to solve a system of linear equations like:

$$\mathbf{Covar}^{-1}(p_s)\, \boldsymbol{y} = \boldsymbol{a}. \quad (6)$$

Therefore, using a system solver reduces extra operation time and storage resources. The covariance matrix has positive-semidefinite and symmetric properties [15], hence, Cholesky decomposition [16] is applicable to this weight solver in regular MV algorithm. Cholesky decomposition is derived from Gaussian elimination, but halves the decomposition operations and is more stable than **LU** decomposition which is the matrix form of Gaussian elimination. $\mathbf{LDL}^T$ decomposition form of the Cholesky decomposition was adopted.

Since the weight solver was iterative, the iterations cannot be parallelized. But inside the iterations, parallelization was achievable. For example, the **L** matrix was formed column by column and only one column in one iteration, but the element calculations within each column of **L** could be parallelized.

Final weight calculation The final step of the weight calculation is to calculate

$$\boldsymbol{w}(p_s) = \frac{\boldsymbol{y}}{\boldsymbol{a}^H\boldsymbol{y}}. \quad (7)$$

As $\boldsymbol{a}$ is a vector of ones, $\boldsymbol{a}^H\boldsymbol{y}$ can be calculated as

$$\boldsymbol{a}^H\boldsymbol{y} = \sum_{n=1}^{L} y_n. \quad (8)$$

Therefore,

$$\boldsymbol{w}(p_s) = \frac{\boldsymbol{y}}{\sum_{n=1}^{L} y_n}. \quad (9)$$

As a result, the final weight calculation step can be parallelized.

3.2.4 Assignment of GPU memory access

The memory access strategy of GPU implementation has an important impact on the overall GPU computational

speed. There are basically three types of memory modules inside the GPU, i.e., global memory, shared memory, and register files. The lifetime of the data in the three memory types is associated to the GPU computing resources respectively. The lifetime of the data in the global memory is associated to the GPU compute grid, the lifetime of the data in the shared memory is associated to the GPU compute block, and the lifetime of the data in the register files is associated to the GPU compute thread.

These three types of memory modules are fabricated according to different architectural hierarchies. The register files reside right next to the GPU processing cores, the shared memory locates in a place with a farther distance to the GPU processing cores, and the global memory locates farthest to the GPU cores. The distance to the GPU cores determines the access speed of the specific memory type. As a result, the access speed of the register files is the fastest, the access speed of the shared memory is slower than that of the register files, and the access speed of the global memory is the slowest among the three types of memory modules. However, on the other hand, the memory size of a specific memory type is inversely proportional to its memory access speed. Thus, the size of the global memory is the largest, the size of the register files is the smallest, and the size of the shared memory is between the size of the global memory and the size of the register files. Therefore, in a GPU program memory assignment, small-size and frequently-used variables can be stored in the register files, while large amount of data should be stored in the global memory. Furthermore, the shared memory is usually used for the shared data space utilized in the computation among the GPU compute threads within a specific GPU compute block.

The GPU memory access assignment of the implementation design is illustrated in Fig. 7. As shown in Fig. 7, the utilization of the slowest global memory was restricted to store input and output data of the beamforming process, i.e., pre-beamform data and post-beamform data. As a result, the communications between ARM and GPU happen only at the beginning and at the end of the MV beamforming algorithm. The faster shared memory was used as storage of intermediate results during the beamforming process. The fastest register files were assigned to hold the temporary results within beamforming steps.

4 Implementation evaluations and discussions

4.1 Experimental setup and simulated scenario

The experimental heterogeneous embedded computing platforms used in our experiments were Jetson TX1 and TK1 evaluation embedded platforms from Nvidia Corporation. Both evaluation boards utilized similar heterogeneous embedded computing platform architecture modules as illustrated in Fig. 3. The major differences between the two evaluation boards were the different GPU module and the different ARM processor module. TX1 platform was using a GPU module of Maxwell GPU architecture and a quad-core Cortex-A57 ARM processor [17], while TK1 platform was using a GPU of Kepler GPU architecture and a quad-core Cortex-A15 ARM processor [18]. The computational capabilities and the power consumption requirements of the two heterogeneous embedded computing platforms were different, which will be shown in Section 4.2.

The experiments in this paper used simulated scenario demonstrated in Fig. 8a to obtain the unaligned medical ultrasonic echo signals for the algorithm implementation. The simulation tool used for the simulated scenario construction was field II simulation tool. Figure 8a illustrates the constructed imaging model for the experiments. There were four cysts in the simulated imaging model, centering at 15-, 25-, 35-, and 50-mm axial imaging depths, respectively. The diameters of the cysts were 6, 6, 8, and 10 mm, respectively. A sample output image of the experiments for the MV adaptive beamforming algorithm was shown in Fig. 8b, which presented the high image quality of the MV adaptive beamforming algorithm and demonstrated the implementation correctness of the MV adaptive beamforming algorithm implementation on the heterogeneous embedded computing platforms.

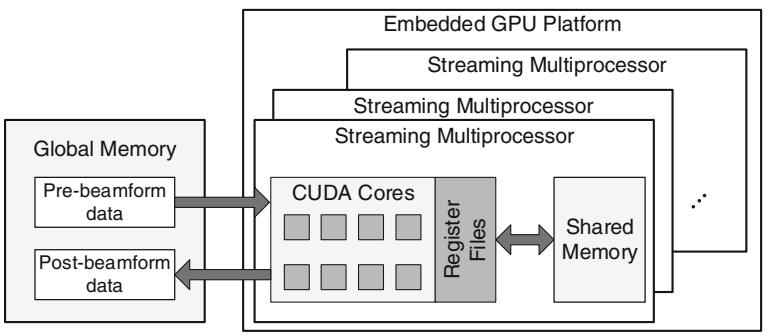

Fig. 7 GPU memory assignment of the implementation design

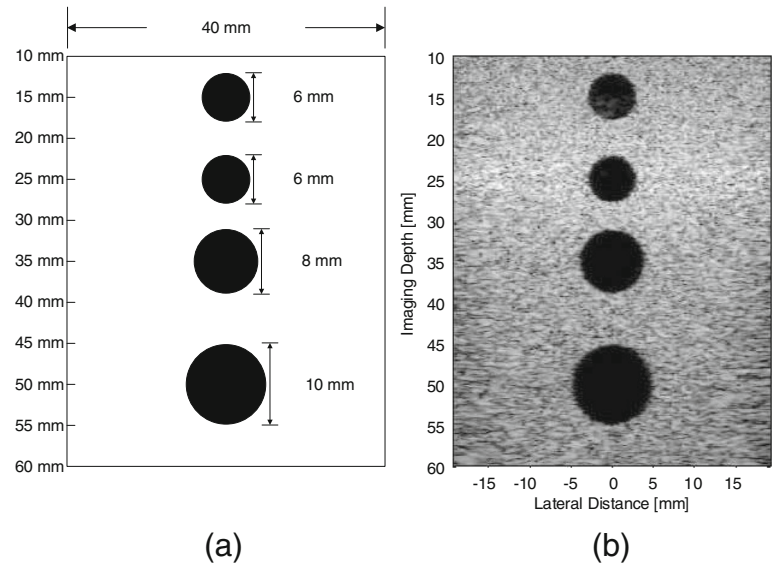

Fig. 8 Simulated scenario used in the experiments for MV adaptive beamforming algorithm implementation. **a** A simulated cyst imaging model. **b** MV adaptive beamforming output image

In order to mimic the real-world medical ultrasonic imaging applications, the Field II simulation tool simulated a 128-element medical ultrasonic transducer with 0.3048-mm element pitch. The pulse repetition rate was set as 5 KHz and the sampling rate was 40 MHz. These parameter settings referred to the real-world medical ultrasonic transducer characteristics. During the simulation of the medical ultrasonic imaging process, the simulation program first simulated the emission process of the medical ultrasonic imaging and then simulated the reception process of the medical ultrasonic imaging. The output of the simulation program was the expected unaligned medical ultrasonic echo signal matrices which were adopted as the input for the MV adaptive beamforming algorithm implementation.

4.2 Evaluations and discussions

4.2.1 Algorithm computing speed

The computing speed evaluation of the MV adaptive beamforming algorithm implementation comprised (a) the giga floating-point operations per second (GFLOPS), (b) the output image frame rate of the algorithm implementation, and (c) the computational speedup of the embedded GPU implementation over its ARM processor counterpart. The algorithm computing speed evaluation was conducted on Jetson TX1 heterogeneous embedded development board. The evaluation test cases included various M and L combinations. While doing the evaluation tests, the value of M was first determined, which was chosen as 16, 32, 64, and 128, respectively. Then, the value of L was chosen from 1 to $M/2$. Therefore, there were 8 M and L combinations when M is 16, 16 M and L combinations when M is 32, 32 M and L combinations when M is 64, and 64 M and L combinations when M is 128. In the experiments, as the number of image pixels, 127×1000 pixels for one image, did not change during the tests, the number of the GPU compute blocks were not changed in the experiments. But the number of the GPU compute threads, a.k.a. GPU compute thread block size, was varying during the experiments. The algorithm computing speed of various GPU compute thread block size was tested so as to find the value of the GPU compute thread block size in the best practices. The GPU compute thread block size test cases covered the numbers from 32 to 1024 which were multiples of 32.

As the values of M and L increased, the computational operations of the MV adaptive beamforming algorithm increased dramatically. Therefore, the larger the values of M and L, the longer the algorithm computational time spent for the implementation. However, the GFLOPS of the implementation might increase when the values of M and L increased. In the experiments, the highest GFLOPS throughput *88.98 GFLOPS* was achieved in the case that M was 128, L was 16, and the GPU compute thread block size was 32, as shown in Fig. 9 using the parallel coordinate plots. In the parallel coordinate plots illustrated in Fig. 9, the leftmost coordinate represented the value of M, the center coordinate represented the value of L, and the rightmost coordinate represented the GFLOPS throughput for the specific values of M and L. Apart from the GFLOPS throughput, the output image frame rate of

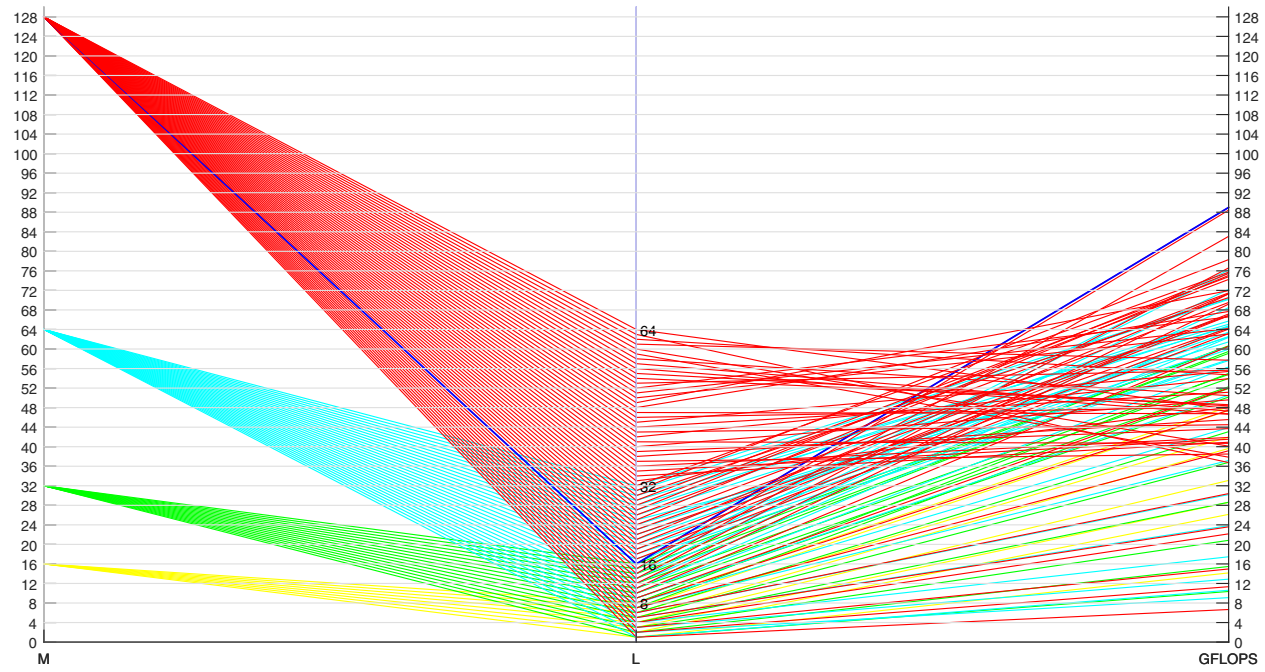

Fig. 9 MV adaptive beamforming algorithm computational GFLOPS throughput of embedded GPU implementation (various *M* and *L*)

the MV adaptive beamforming algorithm implementation was calculated as:

$$\begin{aligned} \text{Frame rate} &= \frac{1}{\text{Time}_{\text{frame}}} \\ &= \frac{1}{\text{Time}_{\text{pixel}} \times \text{Number}_{\text{pixel}}}, \end{aligned} \tag{10}$$

where $\text{Time}_{\text{frame}}$ was the algorithm computational time to obtain one frame of image, $\text{Time}_{\text{pixel}}$ was the algorithm computational time to obtain one image pixel amplitude estimate value, and $\text{Number}_{\text{pixel}}$ was the number of the pixels in one image frame.

In the experiments,the shortest algorithm computational time was **0.017 s** in the case that M was 16, L was 2, and the GPU compute thread block size was 32. Thus, the highest output image frame rate of the MV adaptive beamforming algorithm implementation was $1/0.017 = 58.82$ frame/s (fps). Since the number of image pixels was $127 \times 1000 = 127,000$, the fastest pixel computation time for the MV adaptive beamforming algorithm was $0.017/127,000 = 1.34 \times 10^{-7}$ s. Therefore, when the number of the pixels $\text{Number}_{\text{pixel}}$ changed, the output image frame rate changed, which was expressed as the following: Frame rate $= 1/(1.34 \times 10^{-7} \times \text{Number}_{\text{pixel}})$. The output image frame rate was higher than those in the experiments when $\text{Number}_{\text{pixel}}$ was smaller than 127,000, but the output image frame rate was lower than those in the experiments when $\text{Number}_{\text{pixel}}$ was larger than 127,000. Therefore, the real-time imaging capability of the MV adaptive beamforming algorithm implementation was not only associated with $\text{Time}_{\text{pixel}}$ but also associated with $\text{Number}_{\text{pixel}}$. In the experiments with $\text{Number}_{\text{pixel}} = 127,000$, many test cases for various M and L combinations achieved real-time video imaging frame rate for medical ultrasonic imaging, which was higher than 20 fps.

The computational speedup of the embedded GPU implementation over its ARM processor counterpart was another important evaluation feature of the MV adaptive beamforming algorithm implementation on the high-performance heterogeneous embedded computing platform. The overall speedup results of the GPU implementation over the single-core ARM processor implementation for various M and L combinations were shown in Fig. 10. As seen from Fig. 10, the highest speedup in all the conducted experiments was *102.7 times*, also in the case that M was 128, L was 16, and the GPU compute thread block size was 32. The dark blue lines in Fig. 10 represented the highest speedup cases for the specific values of M. For other specific values of M, the highest speedup for $M = 64$ was 74.4 times when $L = 16$, the highest speedup for $M = 32$ was 50.1 times when $L = 8$, and the highest speedup for $M = 16$ was 31.0 times when $L = 7$. Therefore, as seen from the results, as M decreased, the fraction of L over M increased when the highest speedup was achieved for specific M. In our experiments, L/M increased from 1/8 to nearly 1/2 when M decreased from 128 to 16. The reason of this

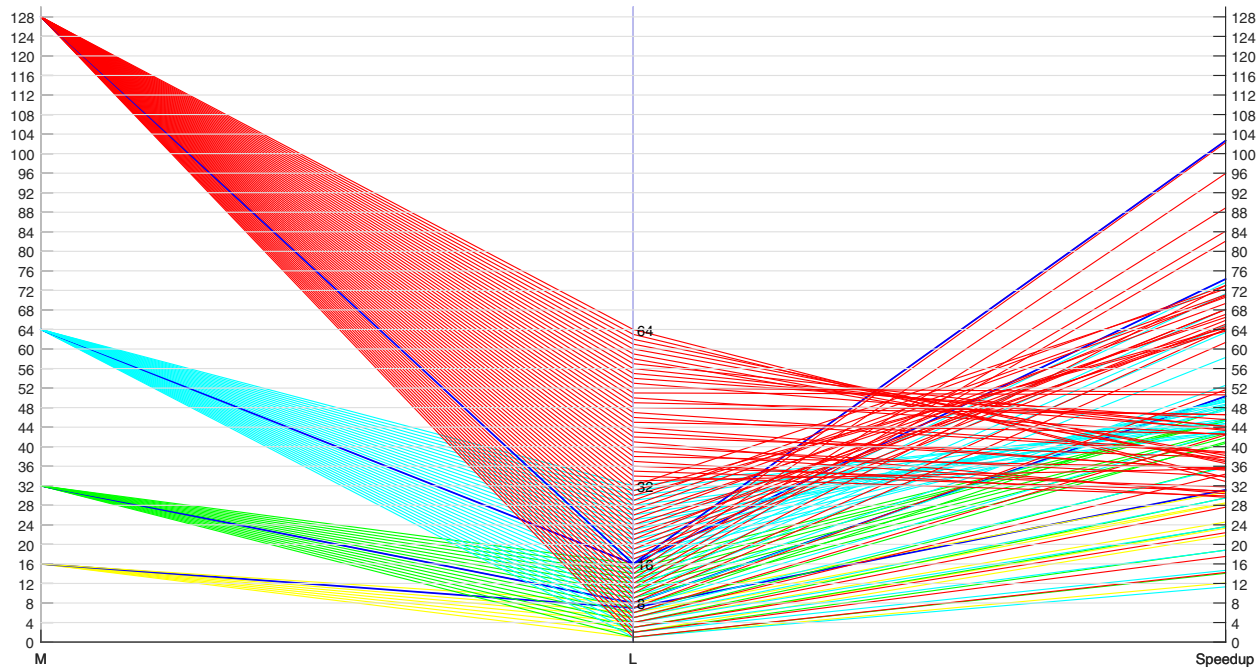

Fig. 10 MV adaptive beamforming algorithm computational speedup of embedded GPU implementation over single-core ARM processor implementation (various *M* and *L*)

trend was because of the GPU compute blocks' matching with the GPU hardware compute cores. When highest speedup occurred, the GPU compute blocks matched with the GPU hardware compute cores' alignment. Furthermore, the computational speedup increased when the computational problem size increased, which means that the high-performance heterogeneous embedded computing platform was more efficient when the computational workload was larger. This was because in our implementation, the hardware/software communications were not frequent. The communications between GPU and ARM only occurred at the beginning and the end of the algorithm calculation. Hence, as the computational problem size increased, the GPU exhibited more powerful computing capability.

4.2.2 Computation energy efficiency

The computation energy efficiency evaluation of the MV adaptive beamforming algorithm implementation included the power consumptions of the embedded computing platforms, and the performance-energy efficiency of the MV algorithm implementation. The performance-energy efficiency of the algorithm implementation was calculated as:

$$\text{Performance} - \text{energy efficiency} = \frac{\text{Computational performance}}{\text{Consumed power}}, \tag{11}$$

where *Computational performance* could be any of the algorithm computing performance evaluation features discussed in Section 4.2.1, but GFLOPS and image output frame rate of the MV adaptive beamforming algorithm were usually adopted. Besides, *Consumed power* was defined as the average power consumption while the embedded computing platform was fully-loaded for the computation.

Two heterogeneous embedded computing platforms, Jetson TX1 and TK1 evaluation boards, were evaluated in the experiments. The power consumptions of idle and fully-loaded embedded computing platforms were recorded to evaluate whether the heterogeneous embedded computing platforms were suitable to be used as portable or mobile diagnostic devices for medical ultrasonic imaging. Table 1 showed the power consumption records for both heterogeneous embedded evaluation boards. As seen from Table 1, the average idle power consumption of TX1 board was lower than that of TK1 board and the average fully-loaded power consumption

Table 1 Average power consumptions of idle and fully loaded embedded platforms

Platform	Idle power (Watt)	Fully loaded power (Watt)
Jetson TX1	2.1	12.2
Jetson TK1	3.3	9.3

of TX1 board was higher than that of TK1 board, which demonstrated the dynamic power saving scheme of TX1 development board. The power consumption measurements demonstrated that the heterogeneous embedded GPU platforms can be used in the portable or mobile diagnostic device constructions.

Referring to (11) and the results discussed in Section 4.2.1, the highest performance-energy efficiency of Jetson TX1 heterogeneous embedded computing platform could be calculated as 88.98/12.2 = 7.29 GFLOPS/Watt or 58.82/12.2 = 4.80 fps/Watt. Besides, the highest operation calculation throughput of the MV adaptive beamforming algorithm on Jetson TK1 heterogeneous embedded computing platform was 37.52 GFLOPS in the case that M was 128, L was 32, and the GPU compute thread block size was 128, and the highest output image frame rate of the MV adaptive beamforming algorithm on Jetson TK1 heterogeneous embedded computing platform was 17.24 fps in the case that M was 16, L was 1, and the GPU compute thread block size was 64. As a result, the highest performance-energy efficiency of Jetson TK1 heterogeneous embedded computing platform was calculated as 37.52/9.3 = 4.03 GFLOPS/Watt or 17.24/9.3 = 1.85 fps/Watt. Therefore, the performance-energy efficiency results of the two heterogeneous embedded computing platform demonstrated that the embedded computing platform consumed more power did not mean that it had a lower performance-energy efficiency. In the experiments, the TX1 embedded computing platform with higher average fully-loaded power consumption exhibited higher performance-energy efficiency.

4.2.3 Platform cost efficiency

One of the important evaluation aspect of the embedded MV adaptive beamforming algorithm implementation was the cost of the implementation platform and its cost efficiency. The cost efficiency was calculated as:

$$\text{Platform cost efficiency} = \frac{\text{Computational performance}}{\text{Platform cost}}, \tag{12}$$

where *Computational performance* could be GFLOPS throughput or image output frame rate of the MV adaptive beamforming algorithm. Furthermore, the cost of the heterogeneous embedded computing platform *Platform cost* was measured in US dollars.

As stated in the official website of Nvidia Corporation, Jetson TX1 heterogeneous embedded computing platform cost *$600* and Jetson TK1 heterogeneous embedded computing platform cost *$200*. When considering the highest computational performance of the heterogeneous embedded computing platforms, the highest cost efficiency of the Jetson TX1 heterogeneous embedded computing platform was expressed as 88.98/600 = 0.148 GFLOPS/dollar or 58.82/600 = 0.098 fps/dollar. Similarly, the highest cost efficiency of the Jetson TK1 heterogeneous embedded computing platform was expressed as 37.52/200 = 0.188 GFLOPS/dollar or 17.24/200 = 0.086 fps/dollar. Hence, the Jetson TX1 embedded computing platform has lower platform cost efficiency as compared to Jetson TK1 embedded computing platform in terms of GFLOPS throughput, and slightly higher platform cost efficiency as compared to Jetson TK1 embedded computing platform in terms of image output frame rate performance.

If the computation energy efficiency and the platform cost combined together, the performance-power-cost efficiency was obtained, which was calculated as:

$$\text{Perforamnce} - \text{power} - \text{cost efficiency} = \frac{\text{Performance-energy efficiency}}{\text{Platform cost}}. \tag{13}$$

As a result, the highest performance-power-cost efficiency of Jetson TX1 heterogeneous embedded computing platform was 7.29/600 = 0.01215 GFLOPS/Watt/dollar or 4.80/600 = 0.00800 fps/Watt/dollar. Similarly, the highest performance-power-cost efficiency of the Jetson TK1 heterogeneous embedded computing platform was 4.03/200 = 0.02015 GFLOPS/Watt/dollar or 1.85/200 = 0.00925 fps/Watt/dollar. Therefore, the Jetson TK1 embedded computing platform has higher performance-power-cost efficiency over Jetson TX1 embedded computing platform in both computing performance measurement means.

5 Conclusions

In this paper, the MV adaptive beamforming algorithm implementation on heterogeneous high-performance embedded computing platforms was investigated. According to the effective high-performance GPU implementation strategies, the MV adaptive beamforming algorithm implementation in high-performance embedded GPU on Jetson TX1 platform can fulfil the real-time imaging requirements for medical ultrasonic imaging in many test cases, and can achieve 102.7 times speedup over its ARM processor counterpart. Besides, the power consumptions of the two experimental heterogeneous high-performance embedded platforms illustrated that the heterogeneous embedded computing platforms can be used as portable or mobile medical ultrasonic devices. Furthermore, the computation energy efficiency and

platform cost efficiency of the heterogeneous embedded computing platforms demonstrated that the heterogeneous embedded computing platforms had relatively good implementation efficiency. As a result, the heterogeneous embedded computing platforms investigated in this paper were suitable to construct real-time portable or mobile high-quality medical ultrasonic imaging devices, especially the Jetson TX1 platform.

Acknowledgements
This work is supported by "Guangdong Natural Science Foundation" (No. 2016A030310412), "Guangzhou Science and Technology Program" (Key Laboratory Project, No. 15180007), and "Guangzhou Science and Technology Program" (No. 201605130108484).

Competing interests
The authors declare that they have no competing interests.

Received: 29 July 2016 Accepted: 3 January 2017
Published online: 23 January 2017

References

1. E Quaia, E Baratella, G Poillucci, AG Gennari, MA Cova, Diagnostic impact of digital tomosynthesis in oncologic patients with suspected pulmonary lesions on chest radiography. European Radiology. **26**(8), 2837–2844 (2016)
2. S Tang, Q Du, T Liu, L Tan, M Niu, L Gao, Z Huang, C Fu, T Ma, X Meng, H Shao, In vivo magnetic resonance imaging and microwave thermotherapy of cancer using novel chitosan microcapsules. Nanoscale Res. Lett. **11**(334), 1–8 (2016)
3. C Love, CJ Palestro, Nuclear medicine imaging of bone infections. Clin. Radiol. **71**(7), 632–646 (2016)
4. BK Hoffmeister, MR Smathers, CJ Miller, JA McPherson, CR Thurston, PL Spinolo, S-R Lee, Backscatter-difference measurements of cancellous bone using an ultrasonic imaging system. Ultrason. Imaging. **38**(4), 285–297 (2016)
5. D Karimi, P Deman, R Ward, N Ford, A sinogram denoising algorithm for low-dose computed tomography. BMC Med. Imaging. **16**(11), 1–14 (2016)
6. JF Havlice, JC Taenzer, Medical ultrasonic imaging: an overview of principles and instrumentation. Proc. IEEE. **67**(4), 620–641 (1979)
7. RSC Cobbold, *Foundations of biomedical ultrasound*. (Oxford University Press, New York, USA, 2007)
8. B Heyde, M Alessandrini, J Hermans, D Barbosa, P Claus, J D'Hooge, Anatomical image registration using volume conservation to assess cardiac deformation from 3D ultrasound recordings. IEEE Trans. Med. Imaging. **35**(2), 501–511 (2016)
9. L Zheng, L Gong, F-C Guo, H Chang, G Liu, Application research on three-dimensional ultrasonic skeletal imaging mode in detecting fetal upper jaw bone. Int. J. Clin. Exp. Med. **8**(8), 12219–12225 (2015)
10. A Bar-Zion, M Yin, D Adam, FS Foster, Functional flow patterns and static blood pooling in tumors revealed by combined contrast-enhanced ultrasound and photoacoustic imaging. Cancer Res. **76**(15), 4320–4331 (2016)
11. J-F Synnevag, A Austeng, S Holm, Benefits of minimum-variance beamforming in medical ultrasound imaging. IEEE Trans. Ultrason. Ferroelectr. Freq. Control. **56**(9), 1868–1879 (2009)
12. Y Li, JA Jensen, Synthetic aperture flow imaging using dual stage beamforming: simulations and experiments. J. Acoust. Soc. Am. **133**(4), 2014–2024 (2013)
13. JA Jensen, NB Svendsen, Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers. IEEE Trans. Ultrason. Ferroelectr. Freq. Control. **39**(2), 262–267 (1992)
14. J-F Synnevag, A Austeng, S Holm, Adaptive beamforming applied to medical ultrasound imaging. IEEE Trans. Ultrason. Ferroelectr. Freq. Control. **54**(8), 1606–1613 (2007)
15. A Papoulis, SU Pillai, *Probability, random variables, and stochastic processes*, 4th edn. (McGraw-Hill, Boston, 2002)
16. GH Golub, CF Van Loan, *Matrix computations*, 3rd edn. (Johns Hopkins University Press, Baltimore, 1996)
17. Nvidia Jetson TX1 developer kit carrier board specification (Nvidia Corporation, Santa Clara, 2016)
18. Nvidia Tegra K1 series processors with Kepler Mobile GPU for embedded applications data sheet (Nvidia Corporation, Santa Clara, 2015)