Performance trends of multicore system for throughput computing in medical application *

Madhurima Pore, Ayan Banerjee,
Sandeep K. S. Gupta
Arizona State University,Tempe, AZ
firstname.lastname@asu.edu

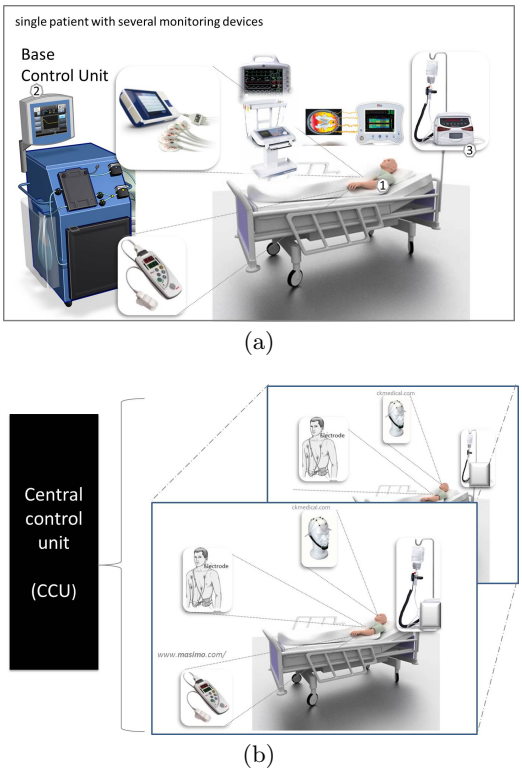
Hari K Tadepalli
Intel
Chandler, AZ
hari.k.tadepalli@intel.com

ABSTRACT

Real time medical control systems (RTMCSes) require high throughput computations to process feedback from physiological monitors and provide real time inputs to actuators for high precision control of physiological state of a patient. The throughput requirement further increases when the RTMCS has to support multiple patients in case of disaster scenarios. Further aggravating the throughput demand is the lack of parallelism in many control algorithms of medical control systems. In this paper, we consider a theoretical study on the throughput obtained from commonly used multi-processing systems such as Graphics processing units (GPUs) and Intel core i7 processors, when subjected to workload comprising of medical control algorithms. We then implement two infusion control algorithms in these systems and experimentally verify the performance trends. Results show that platforms such as GPUs tailored towards increasing performance for parallel implementations may not be suitable for certain types of RTMCSes.

1. INTRODUCTION

Multicore parallel processing computing systems are recently being used for several medical treatment planning applications [6, 9]. Medical applications such as radiotherapy, MRI, CT scan use fast graphics processing units (GPUs) for high performance processing of sophisticated high resolution images or solving complex higher order differential equations for diagnosis and treatment of physiological disorders. Although such high performance applications were intended for treatment planning phase many of these techniques are recently being used for providing real time feedback. For example, in high intensity focused ultrasound (HIFU) the MRI image of the tissue is processed and the temperature rise of the tissue is provided as real time feedback to the controller that controls the intensity of ultrasound. In remote controlled infusion pumps, the drug diffusion dynamics is simulated for 1 hr in the future to provide the future blood glucose concentration as feedback for determining optimal insulin infusion in the present. These real time medical applications need such sophisticated processing to be performed within seconds in order to determine future actions. Thus, they require very high throughput from the multiprocessor computing systems. This paper considers currently existing multi-processor computing systems and analyses their throughput performance for real time medical control applications such as artificial pancreas.



(a)

(b)

Figure 1: Throughput Computing in patient heath monitoring (a) Patient monitoring devices with a base control unit for intercommunication in the devices.[5] (b)Monitoring of multi-patient ICE environment reducing the control devices. Central Control Unit replaces the base control unit in fig.(a)

The recent wave of GPU usage in critical medical applications is inspired by the huge amount of parallelism in the applications. These applications show mainly two types of parallelism 1) Data parallelism 2) Parallelism in multiple patient. Mostly GPUs are used for series of vector operations performed independently on a large number of image pixels.

*This work has been partly funded by NSF, CRI grant #0855527, CNS grant #0834797, CNS grant #1218505 and Intel Corp.

Table 1: Computational requirement of different medical applications

Application	1 patient	Scaled
Epilepsy seizure detection	445 MFLOPs	(100 patients) 43 GFLOPs
Long term monitoring	90 MFLOPs	(100 patients) 9GLOPs
Infusion pump control	1TeraFlops	(10 patient) 10 TFLOPs

This inherent parallelism enables us to allocate operations on one pixel to a single thread in the GPU. A large number of GPUs then evaluate pixels in parallel. The results of the evaluation are then accumulated by a host machine. A real time medical control system (RTMCS) on the other hand may have a different operational mechanism as shown in Figure 1(a). The RTMCS may have several monitoring units and actuation units installed on the patient. The sensors may stream in real time data to a base control unit, which is the main processing unit and can consist of a multiprocessor system such GPU or desktop processor i7. The base control unit runs a controller algorithm, which computes the next actuation input. The actuation is performed by the actuators installed on the patient. The aim is to control the physiological state of the patient by real time actuation such as infusion of drug. The controller algorithm often requires prediction of physiological state in future time for a recent actuation. This requires simulation of multiple instances of human physiological models in a small time interval. Often these predictive algorithms employ Kalman filter type operations, which have to be implemented in a serialized fashion. Thus, although there is significant parallelism in the evaluation of the models of physiology, the predictive unit of the controller introduces significant serial operations. This phenomenon may decimate the benefits obtained from using multi-processor computing systems as base control unit controllers for an RTMCS.

The real time operation of RTMCS necessitate high throughput from the base control unit. Some of the common RTMCS applications such as physiological processing for continuous epileptic seizure detection[1], model based long term monitoring, and infusion pump control system require throughput of the order of MFLOPS to TFLOPS (as shown in Table 1). Further, a single bases station in a hospital may be required to be the controller for RTMCSes from multiple patients. Such a scenario may occur in case of disaster situations such as bombing or natural calamities and may drastically increase the throughput requirement.

As shown in Fig. 1(b), the base control unit gathers data from different sensors, processes the continuous input signal from the sensors, stores it, performs analysis on it and computes the actuation levels for multiple patients. In such an architecture the base control unit is also called the Central Control Unit (CCU). At a scale of thousand patients with different processing requirements such as image processing, fast I/O, high computations, fast data base accesses demands high throughput computing devices to process the real time data. In this paper we consider two types of RTMCS with different levels and patterns of serializations and analyze the capability of commonly used CCUs in terms of their throughput.

2. PROBLEM STATEMENT

We consider two different type of blood glucose control application and evaluate the throughput of different CCUs such as the Intel I7 and the GPU. A glucose control system consists of a controller C that takes the current blood glucose concentration ($G(t)$) as the input and computes the insulin infusion rate at the immediate next time step $t + \delta t$. The controller algorithm C, in order to compute the next insulin level, needs a prediction of blood glucose level at a future time $t + \Delta t$ such that $\Delta t = N\delta t$, where $N \in \mathbf{R}$ and $N \gg 1$. This prediction algorithm is typically a Kalman filter type linear regression model, where the glucose concentration at time $t + \Delta t$ is predicted using Equation 1.

$$G_{pr}(t + \Delta t) = \sum_{i \in S} K_i G_{pr}(t + i\delta t), \quad (1)$$

where, $S \subseteq 1 \ldots N$ and K_is are the coefficients of the linear regression model. Each predicted glucose concentration at a time t is a function of the insulin infusion rate at that time. The function is governed by the physiology of the person and is typically represented using complex physiological models, Phy. These models can be a four dimensional spatio-temporal partial differential equation or an inter-dependent set of time delayed linear differential equations as discussed in Section 4.1. The controller is implemented in the CCU and the entire opertion has to be finished within the small time step δt so that the next infusion rate can be computed in time. Thus, the throughput required from this application is given by $Flops(C)/\delta t$.

The model Phy can have considerable parallelism that can be exploited by using multi-processing. However, each computation of Phy is followed by the computation of Equation 1 and the computation of the control strategy to obtain the infusion rate of the next controller step. This might introduce enough serial computation so that the benefits of parallel computing is subdued. The main problems considered in the paper are -
a) How much speedup is theoretically possible when we implement the medical control algorithms in multi-processing architectures?

b) How do different multi-processing architectures actually perform for different types of control algorithms each having different percentages of serial and parallel operations?

To address the above questions, we implement these algorithms on two types of CCUs, one that uses Intel i7 and the other that uses Nvidia GPU and compare their performance with respect to throughput. We consider two types of glucose control algorithms: a) one that uses spatio-temporal drug diffusion dynamics to predict glucose levels (typically having more parallel code) and b) one that uses a pharmacokinetic model of drug diffusion to predict glucose concentration (mostly serial code). In this paper, we use two definitions of throughput. For a single patient, the throughput of a base control unit is defined as the number of floating point operations per second. For multiple patients the throughput is defined as the number of patients that the CCU can handle before its performance degrades below specific throughput threshold.

3. SYSTEM MODEL

Medical control applications may have many different architectures including several tiers such as hardware devices, applications, data, control, front view, and can be designed for single or multiple patients. The system model considered

in this paper for medical monitoring of patients is shown in Fig. 2. The main components are the sensor devices that continuously feed data to the the central computing unit, CCU. When data is received, the data is preprocessed, then a control algorithm processes the data to compute the future control signals to the actuator. The control signals are sent to the actuators on the patient body. The main aim is to maintain a desirable physiological state of the patient. In this paper, a desirable physiological state is characterized by a specific range of values of the physiological parameters. Hence often the CCU algorithms are range control algorithms as in infusion pump control systems. Further, in multi-patient scenario a single CCU may operate different control algorithms for different patients.

For high throughput requirements in case of multi-patient RTMCS, the CCU has to be optimized for speed up by exploiting the parallelism in the control algorithm C. Let us consider that f is the fraction of parallel code in the controller algorithm. Further, let us assume that the CCU has capable devices to parallelize code and achieve a speedup of X as opposed to a serial implementation of the parallelizable fraction. Performance trends of our highly parallelized spatio-temporal model code (parallelizable upto 98%), for different configurations i.e. gridsize and number of patients, can be estimated using using Amdahl's law. According to Amdahl's law, the overall speedup is given by Equation 2.

$$Sp_{CCU} = \frac{1}{(1-f) + f/X}. \tag{2}$$

To determine X, we need a model of multi-processor architecture that supports parallelism. To obtain a quantitative expression for the speedup X let us consider the parallel architecture of a GPU. A GPU operates in conjunction with a host processor. The host processor can schedule computation to thousands of simple processors in the GPU card as shown in Figure 4. The parallel code can be divided into multiple threads, each of which perform a simple computation. A host machine schedules the threads to a number of stream multi-processors (SM). Each stream multi-processor can process multiple threads at a time. Once all the threads have finished their respective computation, the host machine gathers back the result and continues with the serial part of the controller algorithm. This mode of GPU operation is called single instruction multiple data (SIMD), where each thread perform the same operation on different data points. The data points are organized as a grid of size $G \times G$. The data grid is divided into $B \times B$ blocks and each data point in a block is operated on by a single thread. Hence each block has G^2/B^2 number of threads. Each block can be scheduled to a single SM, while an SM can execute multiple blocks. Typically, an SM has a limit to the number of concurrent threads T_{max} that can be executed in parallel, and a limit on the number of blocks B_{max} that can be executed in parallel. The GPU scheduler schedules blocks to SMs. Thus, if there is a parallel operation on a data grid of size $G \times G$, then the number of blocks that get scheduled on each SM is $SM_n = \lceil \frac{B^2}{B_{max}} \rceil$.

Each SM has a performance curve with respect to the number of thread that they process in parallel. Typically, the flops of an SM is limited by the memory resource of the SM. If the number of memory accesses is within the memory bandwidth then the execution time τ_{exec} of the threads in

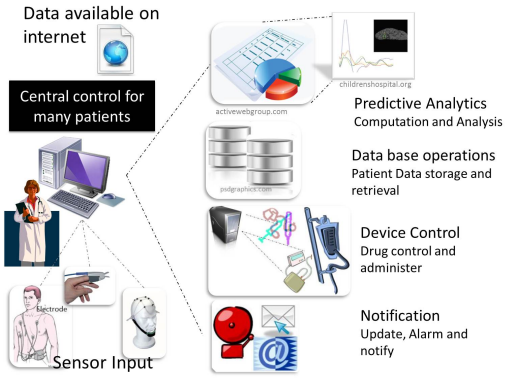


Figure 2: Patient monitoring devices with a central control unit for different functionality.

the SM remains constant as the number of threads increase. However, when the memory bandwidth is fully utilized then the threads stall and their execution time increase. Thus, the Flops obtained from an SM increases with the number of threads (T) and then levels off after the memory bandwidth is fully utilized. Let us consider that the execution time is given by $\tau_{exec} = f(T)$. Since the SM has a limit on the number of threads that can be operated in parallel, if there are more threads than the limit then the threads can be processed in sequential batches of T_{max}. The threads in these batches can be executed in parallel. Thus, the total execution time of the threads is given by Equation 3.

$$T_{thread} = (\lfloor \frac{G^2}{T_{max} \lceil \frac{B^2}{B_{max}} \rceil} \rfloor + 1)\tau_{exec} \tag{3}$$

Once the SMs finish computation, the host machine of GPU needs to gather the results from the SMs and continue with further computation. The SMs communicate the results to the host machine through a communication channel which has a constant delay of τ_c. We assume that there is no collision during the communication of the SM with the host machine and hence we consider that the total communication delay is $SM_n \times \tau_c$. Thus, the total execution time for the parallel code is given by Equation 4.

$$T_{par} = (\lfloor \frac{G^2}{T_{max} \lceil \frac{B^2}{B_{max}} \rceil} \rfloor + 1)\tau_{exec} + \lceil \frac{B^2}{B_{max}} \rceil \tau_c \tag{4}$$

For a serialized implementation of the parallel code the execution time would have been $G^2\tau_{exec}$. Thus, the overall speed up for the CCU is given by Equation 5.

$$Sp_{CCU} = \frac{1}{(1-f) + \frac{f((\lfloor \frac{G^2}{T_{max} \lceil \frac{B^2}{B_{max}} \rceil} \rfloor + 1)\tau_{exec} + \lceil \frac{B^2}{B_{max}} \rceil \tau_c)}{G^2 \tau_e xec}}. \tag{5}$$

4. THROUGHPUT COMPUTING MEDICAL APPLICATIONS

Several types of control algorithms are being currently employed in RTMCSes, including PID, fuzzy logic based, or model predictive controllers. We choose the model predictive controllers since they have a predictive component, which may introduce considerable amount of serial operations in the code. We consider two types of model predictive controllers as described in the following subsections.

4.1 Pharmacokinetic model based controllers

The pharmacokinetic models [10] represent the parts of the body and shows the diffusion process to estimate the drug to be infused into the body. The transient model used here estimates the peripherally administered drug concentration at the cardiac output after a certain time by taking into account different factors such as drug entering different parts of heart and lung tissue, transport delay, recirculated blood, clearance, etc. It involves simultaneously solving the following inter-dependent time delayed differential equations,

$$\dot{x_p} = A_p x_p + B_p(\dot{Q} C_s x_s(t - T_r) + u(t - T_i)), \quad (6)$$

$$\dot{x_s} = A_s x_s + B_s(\dot{Q} C_p x_p(t - T_p)), \quad (7)$$

where, x_p and x_s represent plasma drug concentration in different parts of the body, Q is the drug absorption rate, T_i, T_p, T_r are the drug transport delays, $u(t)$ is infused drug level , and A_p, B_p, A_s, B_s, C_p, and C_s, are constants.

4.2 Diffusion model based controller

This controller has three modes: a) basal, where infusion rate is I_0, b) braking, where infusion rate is a fraction f of I_0, and c) correction bolus, where infusion rate is incremented by I_b. Diffusion dynamics of the drug is spatio-temporal in nature and can be modeled using multi-dimensional PDE Equation 8 [4].

$$\frac{\partial d}{\partial t} = \bigtriangledown(D \bigtriangledown d) + \Gamma(d_B(t) - d) - \lambda d, \quad (8)$$

where $d(x,t)$ is the tissue drug concentration at time t and distance x from the infusion site, D is the diffusion coefficient of the blood, Γ is the blood to tissue drug transfer coefficient, and $d_B(t)$ is the prescribed infusion rate at time t, and λ is the drug decay coefficient. A control algorithm in the infusion pump samples Equation 8 and adjusts the infusion levels so as to achieve the desired physiological effects while avoiding hazards such as hyperglycemia.

The control algorithm hence requires solving the differential equation periodically within the time interval between two control inputs. The Finite Difference Time Domain (FDTD) method is used to solve the differential equations. A double derivative in this method $\frac{\delta^2 V}{\delta x^2}$ can be discretized using the finite difference time domain method [8] and represented as:

$$V^{m+1}(i,j) = K \times [V^m(i+1,j) + V^m(i,j+1) + V^m(i-1,j) + V^m(i,j-1)], \quad (9)$$

where the time is discretized into slots indexed by m, the x and y space dimensions are discretized with slots indexed by i and j, respectively and K is a constant. The entire solution involves computation of each grid point based on the linear combination of neighboring points. For each 5cm×5cm area on the body with a grid size of 1 mm requires a computation of 68 FLOPs per grid point. To solve the equation using this method for 1 hr ahead in the future requires 1 TeraFlops [7].

4.3 Implementation

In this paper, we considered four implemenations: a) parallel implementation of the pharmacokinetic model based controller on Intel i7, b) implementation of the previous algorithm on GTX 680 GPU, c) parallel impementation of the diffusion model based controller on Intel i7, and d) implementation of diffusion model based controller on GPU. The correctness of each of these implementations are tested by comparing an example run with experimental data from

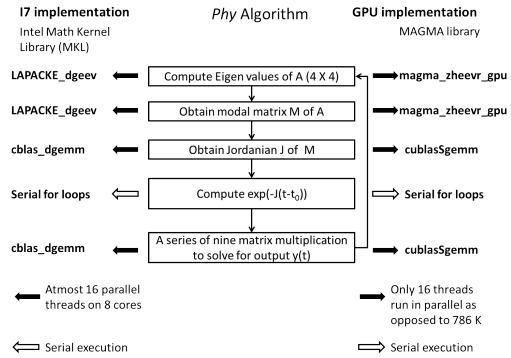


Figure 3: Implementation of pharmaco kinetic model for glucose monitoring on different multicore platforms such as i7 and GPU.

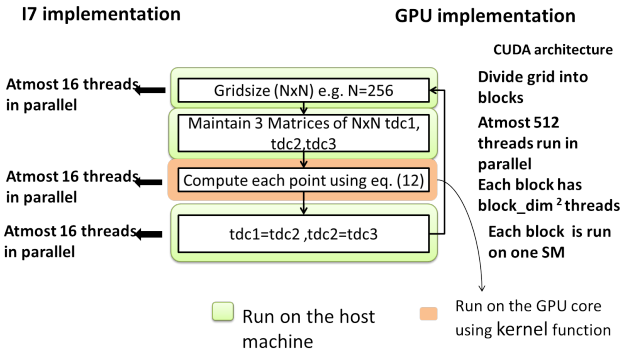


Figure 4: Implementation of FDTD model for chemotherapy on different multicore platforms such as i7 and GPU.

literature [10]. **Exploiting parallelism in computation of multiple patients:** The pharmacokinetic model code does not exhibit much inheritant parallelism. As a result, speedup is obtained by running the multiple patients in parallel. We parallelize the computation for each patient by instantiating multiple threads each for a given patient. In an i7 processor a single core is powerful enough to provide high throughput for the single threaded computations for a given patient. However, a GPU core does not have enough pipelining capability and hence has lower throughput.

The spatio temporal model on the other hand has significant parallelism for a single patient. Hence speed up can be obtained by parallelizing among patients through instantiation of threads for individuals as well as parallelizing the computations for an individual as shown later in this section.

Exploiting parallelism in computation of single patient: The implementation of the physiological model (Figure 3) consists of vector computations such as Eigen value determination of a 4×4 matrix, deriving the eigen vectors to build a modal matrix, and then computing the Jordanian. Parallelized implementation of these operations for the Intel i7 platform are available in Intel math kernel library (MKL)[2] and the specific functions are shown in Figure 3. As this code is highly serial, hardly any speedup can be gained in running a single patient. The vector processing can also be parallelized in a GPU. We use the MAGMA library [3] to implement the vector processing operations in the GPU in a parallelized fashion. Since these operations are on a 4×4 matrix, at most 16 threads can be deployed in parallel even though GPU has the capability of executing 768K threads in parallel. When this code is run on GPU, the over head of transporting the data back and forth to

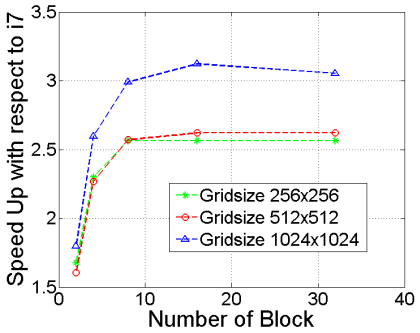


Figure 5: Speedup of Spatio-temporal diffusion control application on GPU for 1 patient

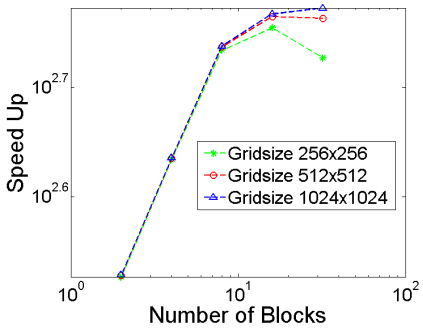


Figure 6: Theoretical speedup computed for Spatio-temporal diffusion control application using equation 5

the host machine and running the code on the less powerful cores result in slow execution.

Although spatio temporal model provides opportunities to parallelize, in an i7 we can only have 16 threads in parallel. Hence, an architecture like GPU overpowers an i7 in this regard. GPU implementation has two parts: a) kernel function, and b) host function. The kernel function computes the drug concentration for each grid point based on Equation 9. In the kernel function a grid point is associated with a unique thread id so that one thread always computes the drug concentration of the same grid point. The host machine first allocates memory in the SMs to store the entire grid of data and then moves the data to the SMs. It then deploys a large number of threads one for each grid point and schedules them on the SMs. Each thread then executes the kernel function in parallel on different grid points. A maximum of 512 threads can be scheduled in each SM. Given that there are 1536 SPs in a GPU, a total of 768K threads running in parallel guaranteeing enormous speed up as seen in §5.

4.4 Theoretical estimation of speedup

In this section we provide detail analysis of our application to estimate the maximum number of FLOPs it can exert given the underlying hardware architecture.

Pharmacokinetic Model on GPU: The pharmacokinetic model has different sections of code such as Eigen Value Computation, Obtaining Modal Matrix, Obtaining Jordanian, etc (shown in Figure. 3) that need to be performed one after another as output of one stage is input to the next state. For example consider the function of Eigen value computation. It involves solving $|A_{4\times4}\text{-}\lambda I|$ and evaluating the function using Newton Raphson method iteratively till solution is obtained. This computation involves series of parallel computation. 1) $|A_{4\times4}\text{-}\lambda I|$ =5 FLOPs forming equation in terms of λ 2) Compute f(x) and f'(x) =9 FLOPs 3) Evaluate the solution. 4) Check if the the evaluated solution reached 0 and repeat Step 1. Total FLOPs =(5+9+1+1). Hence this section of code can exert on an average 16FLOP / 4 cycles i.e 4GFLOPS (as each core has clock of 1.0006GHz).

Spatiotemporal Model on GPU: Kernel computation is iteratively done in spatiotemporal model code. In every iteration, data is transfered from the host to the GPU, computed using kernel code and sent back (shown in Figure. 4) The spatio temporal model kernal requires 73 FLOPs for every grid point. For a grid data (G) of 256×256, each block (B) of 16×16 grid points, we have number of blocks N_B=256 given by $\frac{G\times G}{B\times B}$. In a single time step GPU computes $73\times G^2$ FLOPs.Hence the time requires for computation is 0.495s. The communication time required for $2\times G^2\times 32(Singleprecisiondata)*8(SM)$ at the rate 12.17GBps (observed) is $344\mu s$. This transfer of memory back and forth is needed to be done for 200 ×800 in our case for the iterative loop described in Figure 4 . This requires 55s. As observed, our application set up in the above configuration executes in 87s.

We use the theory discussed in Section 3 to estimate the performance of multi-processor architecture for different block sizes on these two applications. We compute the fraction of parallel code by taking the ratio of the total number of FLOPs for the parallel code to the total FLOPs of the entire code. For the spatio-temporal diffusion model based application we have $f = 0.98$, whereas for the pharmacokinetic model based control algorithm we found $f = 0.22$. The more serial component in the pharmacokinetic model is due to the fact that the differential equations in Equations 6 and 7 are interdependent and cannot be solved in parallel. Based on these fractions we use Equation 5 to compute the theoretical speedup obtained from multi-processor architecture as shown in Figure 6. These graphs are generated by assuming that the execution time of a thread in a SM, τ_{exec}, does not change with increasing number of threads. In reality however with increasing number of threads the execution time also increases. Thus, although the trends of the individual curves will remain the same the difference between the two curves will continue to increase with increasing grid size.

5. EXPERIMENTAL EVALUATION

Setup: The patient monitoring and control unit CCU is run on multi-core system. We use two multi core machines, the GTX 680 GPU platform which has 1536 cores and the Intel i7 with 8 cores to demonstrate the performance of our applications. The GPU cores are much simpler than the i7 cores and are mainly used for SIMD operations. It has a simple pipeline with limited branch prediction and loop unrolling capabilities. It is geared towards fast computation of large number of simple arithmetic operations. On the other hand, the i7 processor is much more sophisticated and supports a more capable pipeline. The parallel programming paradigm of these two units are also different.

In the GPU, the programmer has to take care of partitioning the data into blocks and comunicating it to different stream multiprocessors (SMs). The host processor to the SMs takes this data partition strategy and schedules thread blocks to SMs. On the other hand, in an i7 the programmer can use custom functions, which automatically parallelizes complex

Table 2: Speedup for Pharmacokinetic Model of i7 w.r.t. GPU

Number of patients	Execution Time (i7)	Execution time (GPU	Speedup
1	20	4700	0.0011
2	276	18000	0.0153

Table 3: Speedup for Spatio-temporal diffusion model application on GPU with respect to i7 for multiple patients

# patients	Computation +Communication			Computation Only		
	Exec Time s(i7)	Exec time s(GPU)	Speed up	Exec Time s(i7)	Exec time s(GPU)	Speed up
1	194	87	2.22	14	2	7
2	196	102	1.92	15	4	3.75
4	221	186	1.18	15	6	2.5
8	396	345	1.14	28	18	1.55
16	768	754	1.01	58	35	1.65

mathematical functions and schedules threads.

Evaluation Metrics The medical application is running continuously and captures the real time signals, processes and analyzes them to control the drug infusion rate. The main criterion of performance of these applications is the throughput given by GFLOPS. Some of the applications have a combination of serial and parallel codes and the actual evaluation of these applications can be measured by the number of patients that can be monitored simultaneously for specific input and output signal rates.

Results The pharmacokinetic model application is highly serial application and hence does not perform well on the GPU. The results in Table.2 shows the speedup with respect to GPU i.e execution time on i7/execution time on the GPU for different number of patients. As the number of patients increases, the GPU time increases almost exponentially because most of the code is serial and is run using a single thread on the GPU and due to data being copied to and from the GPU device to the host device. The following experiment shows speedup for the FDTD application using GPU. For a single patient, this application computes the drug level in the tissue of (gridsize/2)mm. As the computation of the entire grid can be broken into different blocks which run on different GPU cores, the speed up is observed to increase with increasing number of blocks. In figure 5, the speedup for GPU with respect to i7 desktop (i.e. execution time on i7/execution time on GPU) is shown by varying the block sizes for different size of the tissue (grid). For a particular gridsize, as the number of blocks increases the curve becomes steady. This is caused due to two reasons, 1) the communication overhead to get the result back on the host machine introduces time delay, 2) GPU has a maximum limit on blocks that can run simultaneously on a SM as well as the total threads that can run in parallel in a SM. For more number of blocks, some blocks may be scheduled in different batches refer §3 . Effect on speedup for multiple patients is shown in the Table 3. For smaller number of patients e.g. 2, GPU performs very good as the computation is highly parallel and spread across different cores. However, as the number of patients increases towards 16 which is equal to number of threads of core i7, the performance of GPU is closer of that of i7. For such a large computation, the GPU performance is limited by the maximum number of threads each SM can run in parallel and the blocks that can assigned to SM for simultaneous execution. As a results the blocks are scheduled in serial batches. Further, due to increasing SMs the communication delay also increases leading to decreased speedup as demonstrated in Table 3. As the communication to the host machine significantly increases the execution time, we get higher speedups considering only the computation time.

6. CONCLUSIONS

In this paper, we have presented a theoretical analysis of the speedup of multi-processor based CCUs in RTMCSes. We have implemented two medical control system algorithms in commercially available multi-processor systems, Intel Core i7 and Nvidia GTX 680 GPU, and experimentally obtained the throughput of these devices. Although data parallelism can provide considerabe speedup, a poor memory organization can result in communication overhead which slows down the application. Hence, to achieve the required throughput, we need to optimize commuication for memory access.

7. REFERENCES

[1] Chen Chiang. Algorithm and architecture design ofepilepsy seizure prediction system. In *Thesis in Graduate Institute of Electronics Engineering, National Taiwan University*, pages 1–97, 7 2011.

[2] Intel Corporation. *Intel Math Kernel Library version 11.0*, 2013.

[3] ICL group at University of Tenesse. *(Matrix Algebra on GPU and Multicore Architectures) MAGMA 1.4 Beta 2*, 2013.

[4] Trachette Jackson and Helen Byrne . A mathematical model to study the effects of drug resistance and vasculature on the response of solid tumors to chemotherapy. *Mathematical Biosciences*, 164(1):17 – 38, 2000.

[5] Kathy Lesh, Sandy Weininger, Julian Goldman, Bob Wilson, and Glenn Himes. Medical device interoperability-assessing the environment. In *HCMDSS-MDPnP. Joint Workshop on*, pages 3 –12, june 2007.

[6] Holger Scherl, Benjamin Keck, Markus Kowarschik, and Joachim Hornegger. Fast gpu-based ct reconstruction using the common unified device architecture cuda. In *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE*, volume 6, pages 4464–4466, 2007.

[7] Sun Shu-Hai and C.T.M Choi. A new subgridding scheme for two-dimensional fdtd and fdtd (2,4) methods. *Magnetics, IEEE Transactions on*, 40(2):1041 – 1044, march 2004.

[8] Allen Taflove and Susan Hagness. *Computational Electrodynamics: The Finite-Difference Time-Domain Method, Third Edition*. Artech House Publishers, 3 edition, jun 2005.

[9] Jeyarajan Thiyagalingam, Daniel Goodman, Julia Schnabel, Anne Trefethen, and Vicente. On the usage of gpus for efficient motion estimation in medical image sequences. *Journal of Biomedical Imaging*, 2011:1:1–1:15, jan 2011.

[10] Rusell Wada and Denham Ward. The hybrid model: a new pharmacokinetic model for computer-controlled infusion pumps. *Biomedical Engineering, IEEE Transactions on*, 41(2):134–142, 1994.