Heterogeneous many cores for medical control: Performance, Scalability, and Accuracy

Madhurima Pore, Ayan Banerjee, Sandeep K. S. Gupta
Impact Laboratory https://impact.asu.edu,
Arizona State University, Tempe, AZ
Email: firstname.lastname@asu.edu

***Abstract*—Many medical control devices used in case of critical patients have model predictive controllers (MPC). MPC estimate the drug level in the parts of patients body based on their human physiology model to either alarm the medical authority or change the drug infusion rate. This model prediction has to be completed before the drug infusion rate is changed i.e. every few seconds. Instead of mathematical models like the Pharmacokinetic models more accurate models such as spatio-temporal drug diffusion can be used for improving the prediction and prevention of drug overshoot and undershoot. However, these models require high computation capability of platforms like recent many core GPUs or Intel Xeon Phi (MIC) or IntelCore i7. This work explores thread level and data level parallelism and computation versus communication times of such different model predictive applications used in multiple patient monitoring in hospital data centers exploiting the many core platforms for maximizing the throughput (i.e. patients monitored simultaneously). We also study the energy and performance of these applications to evaluate them for architecture suitability. We show that given a set of MPC applications, mapping on heterogeneous platforms can give performance improvement and energy savings.**

I. Introduction

Critical patient health monitoring use medical control systems with models of human physiology in a feed back in order to predict and control the drug level in the body. These human physiology models can be computationally complex and can have serial and parallel parts in the application. The parallelism exhibited in such applications has multiple modes: 1) Parallelism due to multiple patients monitored simultaneously (thread level parallelism); and 2) Parallelism within a single application involving prediction using the model (data level parallelism). We hypothesize that when we have such multiple sources of parallelism, heterogeneous platform can run these applications with better performance and lower energy consumption. Heterogeneous platforms have a central unit that can perform highly serial computation very fast and also manage the computation on distributed multi core system.

These medical control applications for critical patient care have several components such as cell phones that are connected to hardware sensors, data collection and processing, applications, front end and storage. The sensor hardware continuously collects the patient data, which is sent to a data center using a cellphone or other communication channel. Remote monitoring of the patient involves continuous monitoring of patients' vital signals such as monitoring patients blood glucose levels, ECG signals and blood pressure. This scenario is described in Figure 1. Once data from different sensors is received, it is processed with different medical applications such as drug infusion controller. Based on sensor data, the controller decides on the level of drug to be infused to the patient's body. These drug infusion controller are often sophisticated and have to be executed for multiple patients in parallel. Further due to the real time requirements of these controller applications, a high throughput is essential to maintain responsiveness for individual patients. At the patient side, the infusion pump adjusts the infusion level of drug based on the decision signal received through the cellphone. Consequently, this paper seeks to evaluate state of the art multi-core platforms in terms of their applicability in these high throughput applications.

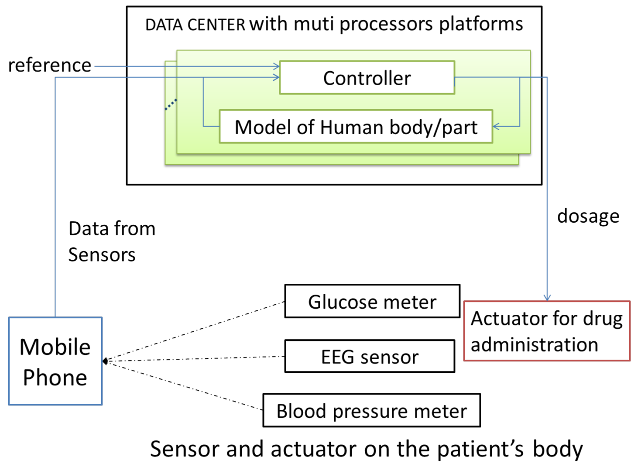
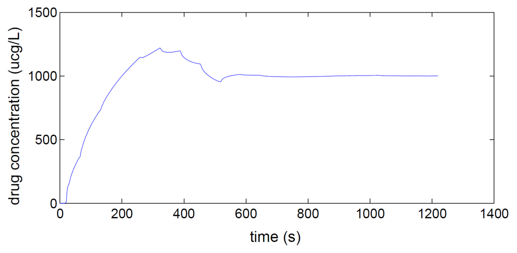
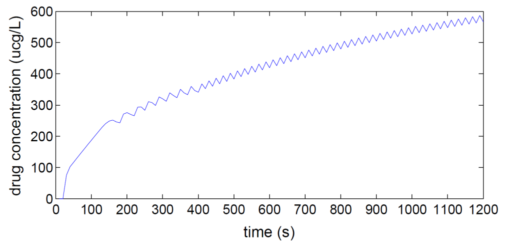


Figure 1: Patient monitoring in hospital and the computationally intensive control algorithm is sent to data center.

The Model Predictive Controller (MPC) systems have a model of process with input and output constraints which is used to predict the output of the model to optimize the



(a) Drug concentration based on Pharmacokinetic model based control.



(b) Drug control using FDTD model of human tissue.

Figure 2: Using Pharmacokinetic model based control system results in fluctuating output concentration as seen in (a) while FDTD based computationally complex model gives more control over the drug concentration as seen in (b).

control signal based on the difference between the reference and predicted output. MPC systems in the medical applications are used to control drug administration for patients [1], [2], [3]. The MPC systems may have different complexity of human models that predict the output with different accuracy. More complex model require larger computation time. However, these applications are time critical because every few seconds the infusion rate is adjusted based on the feedback of predictive model. Thus, MPC based medical applications typically use a very simple Kalman filter type linear predictor models. However, such simplistic predictions may cause fluctuations in the control operation can be severely harmful for the patient. For example, frequent hypoglycemia i.e. an under shoot in glucose concentration may lead to blindness. Figure 2(a) shows MPC system output drug concentration in blood that uses a Pharmacokinetic Model [1]. Figure 2(b) shows usage of a very complex model of human physiology such as Finite Difference Time Domain (FDTD) may require long computation time causing it difficult to meet the high throughput requirement or support the real time operational requirements.

Critical medical applications such as insulin controller using artificial pancreas rely on MPC for accurate real time feedback operation. Given the real time operation requirements, the control algorithm should execute the prediction within a very small time window of the order of 10 s or less. Often the applications require estimation of the human physiology for as long as an hour in the future as feedback [2], [4]. For example, FDTD based spatio-temporal diffusion model of 5 cm$\times$5 cm area on the body with a grid size of 1mm requires a computation of 68 FLOPs per grid point. Hence, predicting drug level 1 h ahead in the future requires computation of 1 Tera FLOPs within 10 s.

The main challenge in computationally complex MPCs is that the computation needs to be completed accurately before the time of drug infusion. Hence, the current many core platforms can be used to monitor several patients at a time. The performance of these applications is demonstrated by meeting the FLOP requirement of the application for individual patient but also the number of patients that are supported simultaneously. We exploit the parallelism of application at thread level and data level and spatial and temporal locality as well as overlapping the computation by communication time of data to match the architecture level parallelism. Main contribution of our work is to match the architecture and application parallelism to extract maximum throughput from the many core systems. We consider three platforms MIC, GPU and Intel core i7 (see Table II) and characterize the performance and energy of our benchmark MPC applications. The i7, which shows lesser energy proportionality performs well for serial types of applications, while GPU and MIC work better for parallel applications. The MPC applications have limitations in achieving the theoretical throughput offered by many core architectures due to inclusion of serial parts. In a hospital environment monitoring multiple patients with different applications, heterogeneous platforms such as MIC, i7 can be more beneficial in terms of performance and energy savings.

The following section discusses our system model used in patient health monitoring. Section III gives the details of MPC controller application. The resource requirements of these applications define the design space parameters. In Section IV, we discuss the resources of the multi core platforms to determine our design space. In Section V, we explore the application level parallelism based on our design parameters and in Section VI, we discuss the details of mapping the MPC application on the multi cores to increase the throughput. We also explore different architecture specific parallelization schemes and compare them in terms of performance and energy savings in Section VII. These evaluation can be used to map different types of applications on specific platforms to yield more energy savings.

II. System Model

In hospitals, patients are continuously monitored for vital signs or physiological parameters such as heart rate, body temperature using different sensors. For critical patients, data collected is used by control algorithms to finely monitor certain physiological parameters such that they do not exceed a predefined threshold value. If the signal is estimated to exceed the threshold, possibly an alarm is generated or the actuator's drug infusion rate is adjusted. Medical devices

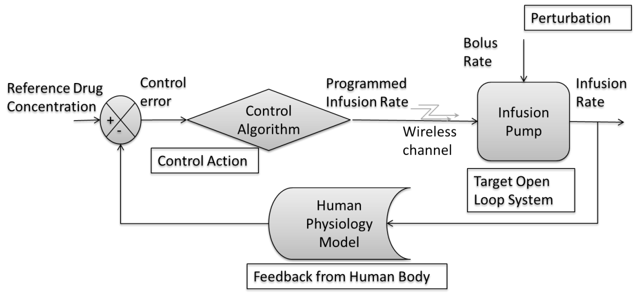


Figure 3: Model predictive controller to maintain the reference drug level in patients' body using human physiological model simulated to give feedback of drug level in parts of patients body based on administered drug.

may use model predictive based controllers to monitor critical patients. Figure 3 shows MPC based system that has a model of human body or part of human body that is used to simulate the infused drug in feedback in order to estimate drug level in different parts of body. The control algorithm then compares it to the reference drug level and decides the infusion rate of the actuator. The sensors and the actuators are close to the patient's body and connected to the MPC through a wired or wireless channel. The simulation of the human body model and the control algorithm run on computationally capable devices such as laptop or servers. However, computational requirements in case of monitoring of multiple patients can be provided by multi-core systems.

III. Throughput computing medical applications

Several types of control algorithms are being currently employed in Medical Control Applications, including proportional-integral-derivative controller 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. The resource requirements of these applications e.g. computational, memory requirements, form the design parameters for evaluation of the multi core platform.

A. Pharmacokinetic model based controllers

The Pharmacokinetic models [1] 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,

$$\dot{x_p} = A_p x_p + B_p(\dot{Q}C_s x_s(t-T_r) + u(t-T_i)), \quad (1)$$
$$\dot{x_s} = A_s x_s + B_s(\dot{Q}C_p x_p(t-T_p)). \quad (2)$$

Algorithm 1 MPC algorithm

```
procedure MPC
    $No\_Steps$ = $Total\_Time/\delta t$                ▷ $\delta$t =time step
    for $i = 1 \rightarrow No\_Steps$ do
        Predict drug concentration $Cp_p$, infusing rate=$x_i$
▷ $\delta$x = increment of x every time step
        Increment Infusion Rate $x_{inc} = x_i + \delta x$
        Predict Future Drug Level $Cp_f$, infusion rate=$x_{inc}$
        Assume drug concentration linearly changes with
infusion rate, slope =$\frac{Cp_f - Cp_p}{x_{inc} - x_i}$
        if $Cp_f > Cp_d$ then ▷ $Cp_d$ is desired drug level
            New Infusion Rate $x_{i+1} = \frac{Cp_f - Cp_d}{slope} + x_i$
        else if $Cp_f < Cp_d$ then
            $x_{i+1} = x_{inc}$
        else
            $x_{i+1} = x_i$
        end if
    end for
end procedure
```

Table I: Variables used in the Pharmacokinetic Model

Symbol	Size	Defnition
x_p	4×1	State variable representing the drug masses in the Cardiopulmonary system for left heart, lung blood, lung tissue, right heart compartments
x_s	4×1	State variable representing the drug masses in the systemic organs such as liver, brain, and muscles, fat tissues and residual tissues
A_p	4×4	State Matrix
B_p	4×1	State Matrix
A_s	4×4	State Matrix
B_s	4×1	State Matrix
C_p	1×4	State Matrix
C_s	1×4	State Matrix
T_s	constant	Sampling delay
T_i	constant	Input delay
T_p	constant	Pulmonary Delay
T_r	constant	Recirculation Delay
Q		Cardiac Output
$\dot{Q}$		Rate of cardiac flow
u(t)		Infusion drug level at time t

These equations can be combined together to give a single equation of the form $\dot{X} = AX + BU$ where, $A = \begin{pmatrix} A_p & B_p\dot{Q}C_s \\ B_s\dot{Q}C_p & A_s \end{pmatrix}$ and $B = \begin{pmatrix} Bp \\ 0 \end{pmatrix}$ and U is the input. Solution for such state space equation is given by $X(t) = e^{A(t-t_0)}X(t_0) + \int_{t_0}^{t} e^{A(t-\tau}B(\tau)u(\tau)d\tau$. We then find the Jordanian J of the matrix At such that $e^{At} = MJM^{-1}$. Hence the next state is given by

$$X = (MJM^{-1})X(0) + (MJM^{-1})tBU. \quad (3)$$

In order to solve the Equation 3 involves Eigen value computation of 36 FLOPs, computation of Jordanian of 54 FLOPs and matrix mutiplications of 233 FLOPs in every iteration to be computed per patient in every 10 s.

1) Resource requirements: Most repeated part of Pharmacokinetic algorithm involves predicting the drug concentration at the sampled point by simulation of the above infusion process with a very fine time granularity, e.g 1 ms. Hence the above Equation 3 is solved to obtain the accurate value of Pharmacokinetic Model which requires the following resources from the underlying platform.

Computation: The Pharmacokinetic model given by Equation 1 and Equation 2 involves solving these two equations in every time step. Each of these equations requires a series of complex mathematical operations (see Equation 3) such as Eigen value of matrix A, Jordanian, modal matrix and series of matrix multiplications in every time step. The maximum size of matrices is 4×4, and output of each operation is input to the next operation. Hence the nature algorithm is highly sequential. Maximally 16 threads can be processed in parallel, hence the CPU resources required can be less.

Memory: As the size of matrices involved is very small (see Table I, the memory requirement is low because of different temporary variables involved in the computation. The variables that are used in one stage of the processing are deleted after providing the required data to the next stage.

Communication: The data requirement of this computation does not require much communication between iterations. At the end of every stage of computation a single floating value or small array is passed on as input to the next stage. Most of the time is consumed in waiting for previous stage input because of sequence of computations one after other.

B. *Finite difference time domain based controller*

Many different fields have applications using FDTD like finite difference simulation of acoustic scattering, numerically model the elastic light scattering by biological cells [5], dispersion characteristics of ferrite-loaded waveguides [6], estimating temperature of human tissue [7]. In this section we discuss about spatio-temporal based FDTD controller that estimates and controls the drug diffused into the human tissue in the time and the space domain. 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 4 [8].

$$\frac{\partial d}{\partial t} = \bigtriangledown(D \bigtriangledown d) + \Gamma(d_B(t) - d) - \lambda d, \tag{4}$$

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 Equation 4, samples drug concentration in the blood and adjusts the infusion pump drug 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

Table II: Multiprocessor platforms for fast processing of medical application

Core Architecture Model No	i7 ivybridge 3770K	GPU GTX 680	XEON Phi 3120P
Core Frequency	3.5GHz	1.006GHz	1.1 GHz
Num Cores	4	1536	57
HW-thread/Core	2	512 (max)	4
Total Last Level Cache	8.192 MB	512KB	28.5MB
Main Memory	16.3815GB	2.048 GB	6GB
Total Power (TDP)	77 W	195W	300 W

two control inputs. The 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 [9] and represented as:

$$\begin{aligned} V^{m+1}(i,j) &= f([V^m(i+1,j) + V^m(i,j+1) \\ &\quad + V^m(i-1,j) + V^m(i,j-1)]), \end{aligned} \tag{5}$$

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. The entire solution involves computation of each grid point based on the linear combination of its neighboring points. For each grid point, a computation of 68 FLOPs are required, hence, for a tissue of 5cm$\times$5cm area with a grid size of 1 mm, to solve the equation using this method for 1 hr ahead in the future requires 1 TeraFlops [10] for every patient expected to be computed within every 10s.

1) Resource requirements: Spatio-temporal diffusion model application has a highly computational algorithm and it involves simulation of all the grid point in order to estimate the drug level in the tissue for every second into the future.

Computation: As discussed above in Equation 5, every grid point involves window type of operation on the grid values of previous iterations. Using the neighboring grid points, new value of the grid is computed. As every grid value involves 68 FLOPs computations, the number of computations increase with the grid size.

Memory: If the entire grid size is available in the cache, the computations can take advantage of the parallel processing in multicore. If the grid data is divided and cached for computation, it needs to combined together after the computation, processed with a few serial operation and then sent back for the grid computation.

Communication: Depending on the location of cached data, data from all the grid points is compiled in order to do serial processing before it is again divided into chunks and sent for processing in parallel. This incurs a lot data transfer overhead in every iteration.

IV. Architecture of multiprocessor platforms

This section describes the details of architectural features of multi core platforms that are exploited for fast processing of different types of parallel applications. We determine our design space based on the resource constraints of each platform. Table II gives a summary of architectural details of the many core platforms used in this study.

A. *Intel(R) Core(TM) i7-3770K (i7)*

Intel i7 [11] is a quad core processor, with hyper-threading, running at a nominal frequency of 3.5GHz with DDR2 -2400, 32GB/4. It has 32KB data cache and 32 KB instruction cache for L1 and 256 KB x 8 way cache for L2. It has 8MB L3 cache and DDR3, 2400MHz RAM of 16GB 4x4 (see Figure 4). The i7 demonstrates very fast performance for memory accesses through different levels of caches. Each quad core CPU can process 8 threads at a time, hence in total 8 threads can be run simultaneously.

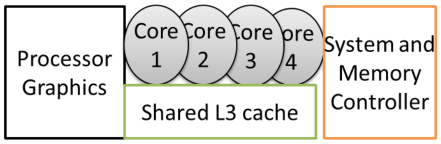


Figure 4: Intel core i7 Ivybridge architecture for parallel computing.

Energy proportionality of i7: The power profile of systems is different with respect to the type of workload. Figure 5 shows the power consumption of the systems at their respective utilizations for medical workloads specifically spatio-temporal diffusion model workload. Power profiling of the i7 involves two parts, first we estimate the peak performance of the system by obtaining the maximum number of grid points computated for spatio-temporal diffusion model application per unit time. Once the peak number of computations is obtained, in the second part, we scale the workload according to the load level. We follow this procedure for all the platforms. As seen in the Figure 5, i7 has comparatively less idle power and power increases almost linearly with the increasing workload.

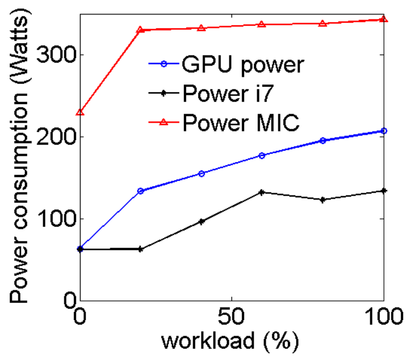


Figure 5: Power profiles of MIC GPU and i7 multi core platforms at different system utilizations.

B. *NVIDIA GPU GTX680 (GPU)*

The GPU GTX 680 [12] (Figure 6 1536 CUDA cores divided in four Graphics Processor Clusters, GPCs, eight next-generation Streaming Multiprocessors (SMX), and four memory controllers. Combined for all the clusters it has has 512KB L2 cache. In all it has 1536 Fused Multiply and Add FMA single precision units and 256 Special Function Units SFU.

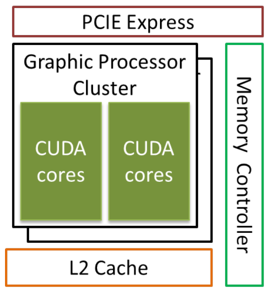


Figure 6: GPU architecture for parallel processing.

Energy proportionality of GPU: The mother board with i7 on the host is fixed with a GPU card. The power consumption is the power of the entire system (mother board with i7+GPU card) with increasing workload. Figure 5, shows the power consumption with increasing load of spatio-temporal diffusion model application. Even though the idle power is less, power consumed increases more rapidly with increasing load.

C. *Intel Xeon Phi Coprocessor 3120P (MIC)*

MIC architecture [13], shown in Figure 7 has 57 cores each associated with L2 cache. There are interleaved Memory controllers and Tag directory connected to the bidirectional ring interconnect for providing fast data from cache. Each core has a complex pipeline architecture that supports 4 threads. The vector processing unit, VPU also supports Fused Multiply-Add (FMA) instructions and hence can execute 32 SP or 16 DP floating point operations per cycle.

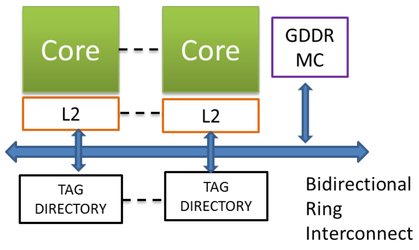


Figure 7: MIC architecture for parallel processing.

Energy proportionality of MIC: The system that has the MIC card in its PCIE slot and host side has Intel core i7. The power profiling is done by measuring the total power (MIC card +i7) when workload MIC is set at different utilization levels. Figure 5 shows that the idle power of the MIC platform is very high. This is due to the large fans and the high power supply unit as well as the idle power of the MIC card. We observe that in this system as the load increases the power consumption is almost constant except for very low loads.

D. Heterogeneous Platforms

The idea of heterogeneous platforms already exists in the multi core systems such as GPU. However, its main purpose has been to divide and distribute the tasks amongst the many cores and collect the results after the computation. Whereas the many core devices, like MIC allows programmer to separate the execution on the MIC cards and the host i7. Also, the high compute cores on the MIC card are provided with a large memory. Hence, the applications that are mostly parallel have a very good performance on the MIC. This leaves the host i7 free for serial types of applications.

V. Parallelism in MPC applications

In this section, we discuss throughput intensive medical applications: spatio-temporal diffusion model application, which can be parallelized effectively on multi cores and Pharmacokinetic model application, which is mostly serial code. In this section we discuss the intrinsics of the applications i.e. computation involved and possible optimizations

A. Exploiting parallelism for a single patient

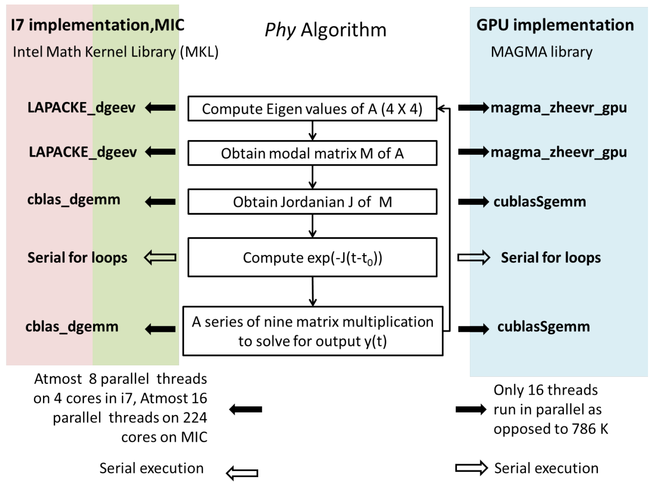

I7 implementation,MIC
Intel Math Kernel Library (MKL)
Phy Algorithm
GPU implementation
MAGMA library
LAPACKE_dgeev
Compute Eigen values of A (4 X 4)
magma_zheevr_gpu
LAPACKE_dgeev
Obtain modal matrix M of A
magma_zheevr_gpu
cblas_dgemm
Obtain Jordanian J of M
cublasSgemm
Serial for loops
Compute exp(-J(t-t0))
Serial for loops
cblas_dgemm
A series of nine matrix multiplication to solve for output y(t)
cublasSgemm
Atmost 8 parallel threads on 4 cores in i7, Atmost 16 parallel threads on 224 cores on MIC
Only 16 threads run in parallel as opposed to 786 K
Serial execution
Serial execution

Figure 8: Implementation of Pharmacokinetic model application on different multi core platforms.

1) Pharmacokinetic Model: The Pharmacokinetic model involves complex mathematical computations (see Figure 8) where output of one stage is input to the next. The iterative part of the algorithm involves computation of Eigen values, build a modal matrix, obtain the Jordanian of A matrix.

Thread level: In these complex operations the matrices involved are of the size 4×4, hence, at most 16 points are computed in parallel. Multiple threads are used to compute these 16 points simultaneously. The next operation cannot start until it receives the output value from the previous operation. Optimized libraries to perform these complex math operations such as Intel Math Kernel Library (MKL) [14] or MAGMA library [15] can be used. However, since there is maximal limit of 16 (as maximum size of matrix is 4×4) for parallelization, on the whole, application throughput is limited.

Data Level: Since the matrices used a small, of size 4×4, the SIMD techniques can be used, but may not be beneficial. We use MKL libraries or MAGMA libraries for the optimization of memory accesses of the array.

Computation and communication overlap: As data size is very small, there is no optimization for computation and communication overlap.

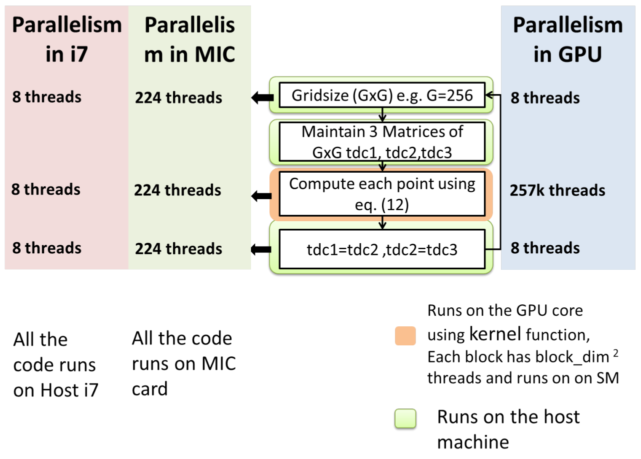

Parallelism in i7
Parallelism in MIC
Parallelism in GPU
8 threads
224 threads
Gridsize (GxG) e.g. G=256
8 threads
Maintain 3 Matrices of GxG tdc1, tdc2,tdc3
8 threads
224 threads
Compute each point using eq. (12)
257k threads
8 threads
224 threads
tdc1=tdc2 ,tdc2=tdc3
8 threads
All the code runs on Host i7
All the code runs on MIC card
Runs on the GPU core using kernel function, Each block has block_dim 2 threads and runs on on SM
Runs on the host machine

Figure 9: Implementation of spatio-temporal diffusion model application on different multicore platforms.

2) Spatio-temporal diffusion model: In the spatio-temporal diffusion model application, the most repeated part is the computation of grid point value using neighboring grid points from the previous iteration. Figure 9 shows the flow of computations involved in this application. For a grid size $G \times G$, all these $G \times G$ points computed in parallel using the previous grid data.

Thread level: A fully parallelized form of this application will involve computing the entire grid using $G \times G$ threads. However, the nature of the algorithm is such that it has few serial processes at the end of each iteration. As a result new updated grid values have to be used for computation in the next iteration.

Data Level: We enable the vector processing to enhance the performance of the sequential accesses in the grid block or the data chunks.

Computation and communication overlap: Some platforms may impose a high overhead of data transfer in order to concentrate the grid data, do serial processes and disperse the data for parallel processing. Computation on GPU is overlapped on the communication to save time. However, the data transfer overhead for GPU platforms is much higher.

For MIC and i7 platforms, all the grid data fits into the memory and memory access overhead over computation is very less.

B. Exploiting parallelism for multiple patient

As discussed earlier, for the Pharmacokinetic model code the parallelism can be obtained by processing multiple patients simultaneously. We instantiate different threads for patients to demonstrate the parallelism in all the three platforms. The spatio-temporal diffusion model code exploits parallelism on patient level as well as on application level for single patient. In i7 implementation, every patient has individual thread. The grid computation is maximally limited by 16 threads. In GPU threads are organized into blocks. Each patient is dedicated a set of blocks. These blocks consist of group of threads that compute the control algorithm for a given patient. Each block of a patient may work with a specific portion of the spatial grid. These blocks are then scheduled on the SMs one after the other. When running on MIC, patient parallelism is instantiated using multiple threads for each patient on the host CPU, also grid point computation is done by dividing the computation amongst the multiple threads on the MIC.

VI. Implementation details

We discuss how these application can be optimized to exploit the resources defined in the design space of each platform in order to improve their performance when run on different multicore platforms.

A. Pharmacokinetic model application

As discussed in Section V, the Pharmacokinetic model involves series of complex math computations. In order to improve the performance we parallelized the computational sections of the applications. However, as a matrix of size 4×4 is involved in each operations, we can use only 16 threads of the underlying platform at a time. The platform specific parallel implementation details are shown in Figure 8.

We use the math library such Intel Math Kernel Library (MKL) [14] for optimized implementation on i7. The i7 cores are powerful, with 4 cores, 2 threads each, they can do the serial computation very efficiently considering the pipeline and cache mechanisms. All the data required in a computation can be easily stored in the cache and deleted once the result is communicated as input to the next stage. At the end of every iteration, the result array is input for next iteration.

In case of MIC we can use the optimized MKL library on the MIC card. However, the invidual cores are comparatively smaller and slower. Hence the serial execution take much longer time for implement on such cores. The data required is easily managed in the cache available for individual core. In spite of having 57×4 available threads, the inherent parallelism of the application is limited to only 16. Hence the rest of the computational resources on the MIC may not be used.

Similarly, GPU code is implemented using the MAGMA library [15]. However, the GPU cores are slower compared to i7 and incur a overhead of data transfer for every iteration. Even though the MAGMA library functions are optimized for performance, the overall the nature of application exhibits highly serial implementation does not exhibit good performance because of the limitation of parallelism exhibited by the application. Due to this, large number of available CPU resources i.e 1536 cores remain idle.

B. Spatio-temporal diffusion model application

The basis of improving the performance of spatio-temporal diffusion model applications is to compute all the grid points at the same time. However in every iteration that involves large parallel processing there is a serial computation involved. The mapping of these grid and serial computations, with platform specific parallelism involved is shown in Figure 9.

In a desktop with i7, each of 4 cores has 2 threads each. Hence, maximally 8 threads can be computed parallel. But the entire grid data is available in the cache and is accessed by all the cores. The fast cache accesses allows a very good performance on i7. Also the serial parts of the application obtain the data from the same cache.

GPU GTX 680, on the other hand has 1536 cores each associated with a smaller cache. A GPU works with the host processor to manage the computation. The architecture of GPU enables us to offload the parallel code onto the card. The data grid is divided into $B \times B$ blocks and each data point in a block is operated on by a single thread. A host machine schedules the threads to a number of stream multiprocessors (SM) by diving a group of threads into number of blocks. The data points are organized as a grid of size $G \times G$ which is divided into $B \times B$ blocks. Hence each block has G^2/B^2 number of threads. These thread blocks are 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. Hence maximum number of blocks that can be scheduled on SM is $SM_n = \frac{B^2}{B_{max}}$. For the GTX 680, atmost 768K threads can be processed in parallel. 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 in every iteration. The communication rates of up to 12.17GB/s are observed for host to device. In our evaluation, we use the grid size of 256×256 with a block of 32×32. The entire grid is transfered onto the GPU card, computed in parallel and the data is sent back to the host machine for serial computation. In the next iteration these updated value of the grid points are used.

In the other case, we avoid data transfer to the host device for every iteration. This results in continuous computation on the grid data without the overhead of communication. The results obtained are not accurate but they give an estimate of performance of a completely parallel application on the GPU as also the overhead of data transfer.

MIC card has on chip L2 cache, supported by individual core L1 cache. The grid data is sent to MIC card memory which is then accessed by the individual cores. The 57 cores on MIC, support 4 threads each with fairly supporting pipeline and cache mechanisms. This allows us to transfer the grid data at the beginning of the processing on to the MIC card and retrieve it after the operation are completed the result value and communicating only single value every iteration. Fast parallel processing by $57 \times 4 = 224$ threads, supporting fast cache accesses using ring interconnect, and capability of processing the serial part of the algorithm on the MIC card improves the performance of the spatio-temporal diffusion model application. Hence this implementation totally avoids grid data transfer to the host device.

VII. Experimental Evaluation

In this section, we evaluate the performance of Pharmacokinetic model application and the spatio-temporal diffusion model application. More accurate prediction of glucose level in the body can be estimated by complex mathematical computations in these algorithms. Main constraint of these control algorithm is the computation time such that glucose level in the body need to be predicted for making the decision of drug level administered. We benchmark both types of application for power, energy and performance on the individual platforms. Based on the results, we demonstrate a sample scenario where combination of applications mapped onto heterogeneous platforms can give more energy savings.
Evaluation Metrics: Within a given time slot, these control algorithm processes and analyzes the input signals to control the drug infusion rate. The performance parameter is throughput given by GFLOPS. Performance of these control algorithm is highly influenced by data transfer overhead to and from the multi core memory. Overall, the number of patients monitored simultaneously defines the performance of a medical application.

Other evaluation metrics we use is the performance of the application per Joule of energy spent. Hence, we account for the total FLOPs for each application or code section. We measure the execution time T_{exec} and power using a power meter as P_{meas}. The MIC and the GPU cards are inserted into the slot with mother board having i7. Hence in order to fairly compare the application's performance per energy for individual platform, we calculate energy as E= $P_{measured} - P_{idle} \times T_{exec}$. Hence our metric is $\frac{FLOPs}{Joule}$.
Measurement Set Up: The MIC card is inserted into the PCIE slots with mother board having i7 processor on the host side. Similarly, we have another i7 desktop with GPU card. The power is measured by inserting the power meter between the Wall plug and the desktop machine. Figure 10 shows the set up of the measurement. Power collector system, requests the current power reading every second and also logs the execution time of the application.

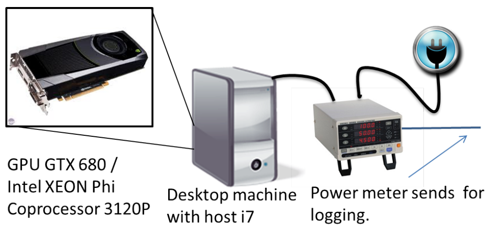


Figure 10: Measurement set up for energy consumption.

A. *Programming Model for benchmark MPC applications*

Pharmacokinetic model application takes input as number of patients that are monitored. The number of threads are generated based on the number of patients. For each patient the application predicts the value of drug level for next half hour. Once the predicted value is obtained, the controller compares this value to the reference drug level and adjusts the signal to the actuator of the infusion pump.
Input to spatio-temporal diffusion model application is the number of patients. Also the size of the tissue (grid size) decides the computation size of the model. In case of GPU, in order to estimate the overhead of data transfer the application can be set to run by avoiding any data transfer. However, in this case the accuracy is compromised. The model is computed to predict the output of drug content in the tissue. Based on this predicted value, the controller compares it to the reference drug level and outputs the signal to the actuator of the infusion pump.

B. *Results*

Table III shows the execution times of half hour simulation of Pharmacokinetic model for different number of patients. As the result indicates, the performance of the Pharmacokinetic model application on i7 is far more superior than GPU or MIC. The reason for such a superior performance is attributed to the fast cores in the i7 and supporting pipeline vector processing compared to very slow individual cores of GPU or MIC. In spite of high computational capability of both GPU and MIC, the other computing resources cannot be utilized because of sequence of operations all involving maximum of 16 threads.

Table IV shows the performance of the control algorithm for grid data size 256×256. As observed, trends indicate the MIC computation is completed fast for more number of patients as the data need not be sent to the host device in every iteration. Platforms like MIC with large cache access and man cores are very suitable for such type of applications that involve both serial and parallel computations. Whereas,

Table III: Performance of Pharmacokinetic Model on multi core platforms.

Number of patients	Execution Time (GPU)	Execution time (MIC)	Execution time i7
1	213	84.85	5.58
2	992	89.02	6.26
4	2081	94.92	7.48
8	2584	118.36	13.11
16	3552	240.73	24.78
32		2066.99	53.86
64			336.81

for GPU the data transfer overhead is much higher as indicated by comparing the (computation only) vs. (computation + communication). The last two column indicate that if the applications have only computation, but the memory accessed involve neighboring data points in the computation (e.g. a window operation for each grid point), the execution time is almost double. Performance for the i7 is reasonable for smaller number of patients however as the number of patients increases, the computational time increases rapidly.

Table IV: Performance of Spatio-temporal diffusion model application on i7 MIC and GPU for multiple patients.

Gridsize 256×256					
	i7	MIC	GPU		
#patients	(s)	(s)	with communication (s)	computation only w/ neighbor (s)	computation only w/o neighbor (s)
1	10.21	2.7	83	3	1
2	80.86	8.12	94	6	2
4	82.22	8.29	293	12	5
8	82.4	135	541	24	9
16	83.83	136.2	993	47	19
32	176.05	142.01	1953	95	36
64	762	193	3823	190	73

The energy and the performance results for the Pharmacokinetic model for different code parts is shown in Table V. The serial nature of the code makes use of the powerful cores in i7 incurring very less time in the data communication and also faster computations. The FLOPs /Joule for i7 also indicates that i7 will be more useful for such applications involving complex math computation but lesser degree of parallelism. The spatio-temporal diffusion model code seems to utilize the computation capability of the multi core platforms. Considering the Table VI, if 64 patients are being monitored simultaneously, almost 45 TFLOPs need to be computed before the infusion time. The subsequent columns show the energy consumption and the performance of application on different platforms. As seen in Table VI, MIC gives better performance for our application considering the FLOPs/Joule are much higher whereas GPU and i7 give lower FLOPs/Joule considering the serial parts in the code involves large communication overhead and computation capacity respectively.

C. Heterogeneous computing

As seen in Section VII-B, the spatio-temporal or the Pharmacokinetic application have a different performance and energy consumption when run on different multicore platforms. The results indicate that Pharmacokinetic application will run more efficiently and fast on i7 while the Spatiotemporal application performs better on the MIC. In the real hospital scenario, different patients may be monitored with different application based on the criticality and need of the patient. In case of different applications that are monitoring multiple patients at a given time, can we utilize the existing heterogeneity of platform e.g. i7 with MIC card to gain more energy savings and better performance?

Consider a scenario, where we have a sample set 3 patients that are monitored using the Pharmacokinetic model application and 16 patients are monitored using the Spatiotemporal diffusion model application. This sample set of applications is running simultaneously on different platforms. Figure 11 shows plot of total execution time and energy consumption of our sample application set when running on the platform. Note that the energy consumption shown is inclusive of the idle energy. As Pharmacokinetic application performs well on i7 and Spatiotemporal application perform well on the MIC we use this combination. The last value in Figure 11, indicates the use of MIC-i7 gives better performance and energy savings.

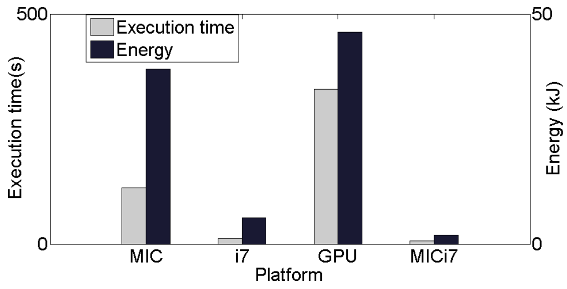


Figure 11: Sample run showing energy and performance of 3 Patients monitored by Pharmacokinetic Model and 16 patients monitored by Spatiotemporal Model Application.

VIII. Conclusions

In this paper we provide a experimental evaluation of different types of medical control algorithms for throughput processing on many core platforms. When subject to serial types of computation, the desktop processing like i7 processors will provide enough resources to meet the throughput requirement. This is demonstrated by the performance of Pharmacokinetic model applications. The application performance is poor on platforms like GPU or MIC. Some control algorithm allows us to exploit high parallelism, such as Spatiotemporal model application, the performance depends on the data transfer overhead incurred

Table V: Performance of Pharmacokinetic model application run on i7, MIC and GPU for multiple patients.

sample: Number of patients =16							
		Energy(kiloJoule)			Performance (MFLOPs/kJ)		
function	FLOPs	i7	MIC	GPU	i7	MIC	GPU
eigen	36	0.542	3.50	125.9	66.55	10.29	0.2864
mmul	286.7	1.080	14.11	503.7	265.4	20.32	0.5692
datatransfer		1.145	130.71	1534.0			

Table VI: Energy and Performance/joules of spatio-temporal model application on i7, MIC and GPU for multiple patients.

sample: Gridsize 256×256							
		Energy(kJ)			Performance (TFLOPs/kJ)		
#patients	TFLOPs	i7	MIC	GPU	i7	MIC	GPU
1	0.713	0.27	0.636	5.751	2.65	3.95	0.1240
2	1.426	4.10	0.636	8.071	0.347	3.86	0.1767
4	2.852	5.37	0.695	17.12	0.53	4.101	0.1666
8	5.704	7.48	12.21	33.27	0.762	0.467	0.1715
16	11.41	11.28	13.82	64.18	0.8308	0.826	0.1777
32	22.82	24.88	17.28	128.4	0.92	1.32	0.1777
64	45.63	107.90	28.06	259.2	0.423	1.626	0.1760

in case of huge data sets as observed in GPU devices. Due to combination of serial and parallel parts in such applications involving large data sets, multicore platforms with huge cache such as MIC have better performance. We compare the performance per Joule of the applications to find best mapping of application on to the platform. The results show that with suitable mapping decisions such as ones using heterogeneous platforms we can gain more performance and energy savings.

Acknowledgement: This work has been partly funded by NSF CNS grant #1218505 and Intel Corp.

References

[1] R. Wada and D. Ward, "The hybrid model: a new pharmacokinetic model for computer-controlled infusion pumps," *IEEE Transactions on Biomedical Engineering,*, vol. 41, no. 2, pp. 134–142, 1994.

[2] M. Wilinska, J. Allen, D. Elleri, K. Kumareswaran, M. Nodale, M. Evans, D. Dunger, and R. Hovorka, "Overnight closed-loop insulin delivery in patients with type 1 diabetes," *5th European Conference of the International Federation for Medical and Biological Engineering, Springer Berlin Heidelberg*, vol. 37, pp. 961–963, 2012.

[3] M. Pore, A. Banerjee, S. K. Gupta, and H. K. Tadepalli, "Performance trends of multicore system for throughput computing in medical application," *2nd Workshop on Performance Engineering and Applications at HIPC*, Dec 2013.

[4] S. L. Shafer and K. M. Gregg, "Algorithms to rapidly achieve and maintain stable drug concentrations at the site of drug effect with a computer-controlled infusion pump," *Journal of Pharmacokinetics and Pharmacodynamics*, vol. 20, pp. 147–169, 1992, 10.1007/BF01070999.

[5] R. S. Brock, X.-H. Hu, P. Yang, and J. Lu, "Evaluation of a parallel FDTD code and application to modeling of light scattering by deformed red blood cells," *Opt. Express*, vol. 13, no. 14, pp. 5279–5292, Jul 2005.

[6] J. A. Pereda, L. A. Vielva, A. Vielva, Miguel, A. Vegas, and A. Prieto, "FDTD analysis of magnetized ferrites: application to the calculation of dispersion characteristics of ferrite-loaded waveguides," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 43, no. 2, pp. 350–357, 1995.

[7] L.-K. Wu and W. Nieh, "FDTD analysis of the radiometric temperature measurement of a bilayered biological tissue using a body-contacting waveguide probe," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 43, no. 7, pp. 1576–1583, 1995.

[8] T. Jackson and H. Byrne, "A mathematical model to study the effects of drug resistance and vasculature on the response of solid tumors to chemotherapy," *Mathematical Biosciences*, vol. 164, no. 1, pp. 17 – 38, 2000.

[9] A. Taflove and S. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method, Third Edition*, 3rd ed. Artech House Publishers, jun 2005.

[10] S. Shu-Hai and C. Choi., "A new subgridding scheme for two-dimensional fdtd and fdtd (2,4) methods," *IEEE Transactions on Magnetics,*, vol. 40, no. 2, pp. 1041 – 1044, march 2004.

[11] "Intel website for i7," http://www.intel.com/content/www/us/en/processors/core/core-i7-processor.html, Oct. 2013.

[12] "GPU GTX 680," http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-680/specifications, Oct. 2013.

[13] "Intel website for MIC," http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html, Oct. 2013.

[14] "Intel math kernel library version 11.0," http://software.intel.com/en-us/intel-mkl, 2013.

[15] "ICL group at University of Tenesse, (Matrix Algebra on GPU and Multicore Architectures) magma 1.4 beta 2," http://icl.cs.utk.edu/magma/, 2013.