Heterogeneous Concurrent Execution of Monte Carlo Photon Transport on CPU, GPU and MIC

Noah Wolfe, Tianyu Liu, Christopher Carothers and Xie George Xu
Rensselaer Polytechnic Institute
Troy, NY 12180
Email: wolfen@rpi.edu

***Abstract*— In this paper, a new level of heterogeneous concurrent execution of Monte Carlo photon transport is presented. ARCHER, an application for computing radiation dosimetry for CT imaging involving whole-body patient phantoms has been extended to execute on any combination of CPUs, GPUs and MICs concurrently. The goal is for ARCHER to detect and simultaneously utilize all CPU, GPU and MIC processing devices available. Due to the irregular nature of the Monte Carlo photon transport algorithm, a new "self service" approach to organizing the heterogeneous device computing has been implemented. This approach efficiently and effectively allows each device to repeatedly grab portions of the domain and compute concurrently until the entire domain has been simulated. New timing benchmarks using various combinations of various Intel and NVIDIA devices are made and presented. A speedup of 13x has been observed when utilizing Intel's Xeon X5650 CPU, Intel's Xeon Phi 5110P MIC and NVIDIA's K40 GPU concurrently versus just the Intel Xeon X5650.**

Keywords—ARCHER; CPU; GPU; MIC; Heterogeneous; Concurrent Execution;

I. INTRODUCTION

Currently, the world of high performance computing is in a state of constant change. As technology advances, computing architectures continue to evolve as well. The rapid development of heterogeneous computing systems is providing easy access to many computing architectures. While a very interesting topic in itself, this paper does not seek to promote one single architecture. Instead, this paper focuses on the irregular development for the simultaneous utilization of all computing architectures on a combined CPU, GPU and MIC platform. With heterogeneous computing, one isn't limited to a single architecture/development platform anymore.

In the medical field, this can mean using a hardware system's entire computing capability to generate simulated patient CT dose values in the doctor's office in real-time. Some early research in medical physics implemented a complex and irregular Monte Carlo transport method on the CPU, GPU and MIC platforms to simulate the CT scan procedure and calculate radiation dose [6]. The codes are separately developed and tested and optimized on each platform. The significance of the heterogeneous computing method introduced in this paper is to maximize the system usage and increase patient throughput.

This all goes to show that as computing architectures advance, the hardware trend is to pair up architectures and increase the computational power. The challenge remains to generate programs that utilize these irregular computing systems concurrently. This paper contributes to that effort by presenting a new implementation of concurrent execution on CPU, GPU and MIC. The first of its kind in Monte Carlo radiation transport, ARCHER-CT now dynamically detects and concurrently utilizes all computing devices in heterogeneous computing systems. Early results show 5x, 10x and 13x speedups respectively when adding an Intel MIC, NVIDIA K40 GPU, and both to an Intel Xeon CPU.

The ARCHER project is funded by the National Institute of Biomedical Imaging and Bioengineering (NIBIB) (R01EB015478)

II. BACKGROUND

A. Computing Architectures

In this paper, three of today's prominent computing architectures are discussed. These architectures consist of the CPU, GPU and MIC. In today's computing world, these three computing architectures are readily available to developers seeking high performance computing.

1) CPU: A common computing architecture for quite some time now, the CPU is typically one physical chip consisting of several CPU cores. These CPU cores are completely independent with their own execution pipelines, registers and control units that allow them to execute application instructions in parallel. Many of today's CPU processors also have the ability to perform multiple threads per core. Although having multiple threads per core does not necessarily translate to linear speed-ups within each core, it does allow each core to fully utilize its execution pipeline.

2) GPU: A rapidly advancing architecture, the GPU is a very powerful computing option. Used as an accelerator, the GPU is attached to the host system via PCIe. Each GPU card contains a certain number of Streaming Multiprocessors (SMs). These SMs are composed of a given number of lightweight Compute Unified Device Architecture (CUDA) cores that can each perform one instruction on separate data. Execution on the SMs is done using blocks and each block is composed of a given number of threads. The number of blocks and threads per block are set and controlled by the user. Each block of threads, when ready, gets assigned in warps (groups of 32 threads) to a group of 16 cores within an SM for computing. All threads in the same warp follow Single Instruction Multiple Thread (SIMT) execution [3]. They perform the same operations at the same time on different data. As long as conditionals and divergence among threads in

978-1-4799-7056-8/14 $31.00 © 2014 IEEE
DOI 10.1109/IA3.2014.11

a block executing on an SM are kept to a minimum, an application can achieve impressive speedups.

3) MIC: The Intel MIC architecture combines many lower powered CPU cores onto one chip connected via a bidirectional ring interconnect. With a frequency of above 1 GHz, each core supports 4 hardware threads. To go along with the increased number of cores and threads, the MIC architecture also has wider vector units than the CPU. Wider 512-bit vector units matched with a new AVX-512 SIMD instruction set, mean it can perform up to 16 single-precision floating-point operations per clock cycle [1]. Intel's first iteration Knights Corner Phi coprocessors provide 57 to 61 cores in the form of a PCIe accelerator card.

B. ARCHER-CT

ARCHER-CT is a simulation tool created for quantifying and reporting patient-specific radiation dose for x-ray Computed Tomography (CT) scan procedures.

1) Irregular Compute Process: Following the Monte Carlo particle transport method, ARCHER traces photon particle paths from creation to when they are either absorbed or leak out of the range of interest. This complex behavior simulates on the order of 100 million random particles in a large 3D patient phantom domain composed of over 5 million voxels. The compute process is highly irregular because in the random walk process, each random particle can progress through any arbitrary number of voxels and no two particle paths through the domain are exactly the same. Fig. 1 shows some normal history paths of particles in Monte Carlo simulation. The free path distance D and scatter angle θ are randomly generated with appropriate distributions. The unpredictable and divergent nature of this transport method makes it very hard to extract SIMD parallelism.

These particles are emitted in scan rotations along the height of the phantom indicated by a scanRotationIndex value. Each axial scan is simulated independently with a given number of particle histories. The parallel strategy takes advantage of independence of individual histories computed in the transport process and requires less inter-processor communication than other methods. This approach is considered massively parallel as no particle history depends on any information from any other particle history. The highly parallel nature of this simulation makes it a great candidate for heterogeneous concurrent execution.

III. Program Implementation

A. Device Organization

As shown in Fig. 2, each computing architecture is connected to one another via the PCIe bus. The CPU is the main computing unit located in the CPU socket and the GPU and MIC devices are accelerator cards connected through the PCIe slots.

Working with the GPU device requires the CPU to act as a servant. The CPU is needed to initialize memory, transfer data and submit kernel jobs to the GPU device. In order to eliminate wasted CPU and GPU compute time spent servicing and waiting to be serviced, we dedicate one CPU thread via a Message Passing Interface (MPI) rank R0 to control the GPU. The remaining CPU threads are grouped together in a separate

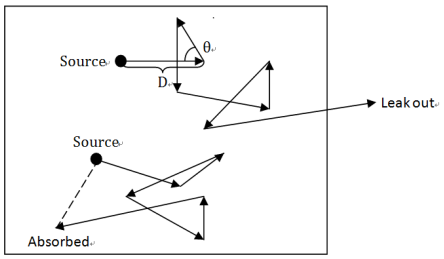

Fig. 1. ARCHER-CT irregular compute process diagram.

MPI rank R1 to compute on their own. Shown in Fig. 2, this method allows both CPU and GPU to efficiently compute concurrently.

For the MIC, Intel describes three "compute modes" or "execution models" for running MPI programs. For concurrent execution, we use the "native" execution model. In this case, each MIC device is seen as extra many-core compute node that runs the simulation on its own micro Linux operating system.

B. Self-Service Compute Structure

Scheduling and balancing work loads among different devices and different architectures becomes very important to reduce the idle time. To do so, a new "self-service" technique is implemented that allows each separate device to grab an appropriate amount of the domain to compute. In the case of Monte Carlo particle transport, the computational domain is the set of scan rotation index values with n-many particles to simulate per index. These scan rotation index values are shown as the red rings in Fig. 2. The transport execution is setup so that each device grabs however many scan rotation index values they need for saturation, computes the particle paths for all particles histories within those index values, and then repeats the process until there are no more index values left.

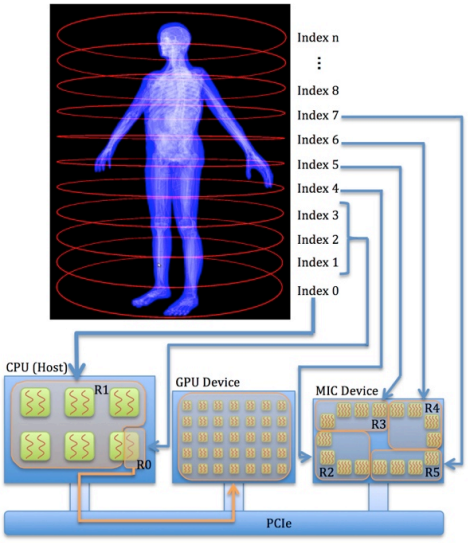

Fig. 2. 3D patient phantom domain distribution and device layout.

C. Device Synchronization

In an attempt to make sure all index values are computed and to prevent repetition of a scan index, MPI is used to pass around the current rotation index value. Each MPI rank has a local copy of scanRotationIndex. This variable starts at 0 and tells which index is to be computed next. Before each rank selects an index value, they must probe to see if there is an incoming message (an updated index from another rank). If so, it receives all the messages and updates the local index accordingly. Then the rank increases the local index by however many index values it will compute and then sends the new local index to all other ranks. This process is repeated until all index values are computed.

IV. Program Composition

In order to have one simulation work on multiple different devices concurrently, multiple code sets and applications are compiled separately and executed together. Here the functionality along with compilation and execution are described.

A. System Device Query

One major new feature for the ARCHER program is the ability to detect the underlying compute architectures and dynamically select and run concurrently on those devices. A separate program has been created using the Open Computing Language (OpenCL) and CUDA to detect the underlying architectures and their corresponding computational specifics. Based on the query result, the user can either select which of the available system architectures to use or select all. After the selection process, the program outputs all relevant computing information to file. This information includes everything from processor frequency to number of cores and threads per device. This data will be later used to generate the optimal data distribution and execution.

B. $ARCHER_{CPU\_GPU}$

This version of the ARCHER transport computation application is a subset of code that is tailored and optimized for efficient and effective concurrent execution on both the CPU as the host and one or more GPUs as devices. The CPU rank computes the transport simulation using openMP to utilize all threads. To overcome the divergent irregular nature of Monte Carlo transport, the GPU runs multiple streams. Each stream is another instance of the compute kernel running with a different scan index. This allows the GPU to be fully saturated. Using the Self-Service compute structure, both the CPU and GPU compute concurrently.

C. $ARCHER_{CPU\_MIC}$

$ARCHER_{CPU\_MIC}$ is tailored and optimized for efficient and effective concurrent execution on both the CPU and one or more MIC devices. This time, the CPU has just one MPI rank and uses OpenMP to compute the transport simulation using all threads. As previously mentioned, $ARCHER_{CPU\_MIC}$ has been setup to run following the "native" execution model. This allows the MIC to run multiple instances of the exact same code as the CPU. The only difference is in the compilation step where we add the flag -mmic to specify that we are compiling and linking an executable for the MIC architecture. With a CPU compiled executable residing on the host CPU and a MIC compiled executable of the exact same code residing on each MIC device, Intel MPI management tool mpirun is used to launch both executables to compute concurrently.

D. $ARCHER_{CPU\_GPU\_MIC}$

To achieve concurrent execution across all three architectures, a composition of both $ARCHER_{CPU\_GPU}$ and $ARCHER_{CPU\_MIC}$ code sets are compiled and deployed to each end device separately in a similar manner to the $ARCHER_{CPU\_MIC}$ case. The first compiled executable, based on $ARCHER_{CPU\_GPU}$, is created for the host CPU to run, controlling GPUs and computing with the remaining CPU resources. The second executable is based on $ARCHER_{CPU\_MIC}$ and is created for the MIC to run and compute concurrently with the CPU and GPU. The mpirun utility is again used to generate the corresponding number of MPI ranks running the host executable on the host and the corresponding number of MPI ranks running the MIC executable on the MIC.

V. Computing System

In the interest of a broad and fair comparison, the new heterogeneous ARCHER implementation has been tested on a number of devices as shown in Tables 1 and 2. The computational specifics of each device that most affect the speed of the simulation are presented along with release date and introductory price.

A. Intel CPU and MIC Devices

The ARCHER simulation is tested on a newer Intel Core i5-4430 desktop CPU and an older midrange Intel Xeon X5650 server CPU. The MIC architecture is much newer than the CPU so device selections are much more limited. As a result, only one device, the Intel Xeon Phi 5110P coprocessor, is tested.

B. NVIDIA GPU Devices

Many NVIDIA GPUs have been benchmarked. First is a very popular and common desktop GPU, the NVIDIA GeForce GTX 670. The second GPU is an older compute dedicated NVIDIA Tesla M2090. The last two GPU devices are the new top of the line, compute dedicated NVIDIA K20 and K40.

VI. Results

Each timing result includes the transport computation time and does not include the file IO and device initialization time. Also, all timing results are computed on each rank using the instruction RDTSC. This machine level instruction returns the number of cycles the processor has completed since last reset.

TABLE I. Intel CPU and MIC Device Comparison

Product	Intel Core	Intel Xeon	Intel Xeon
Model	i5-4430	X5650	5110P
Microarchitecture	Haswell	Westmere	Knights Corner
Cores	4	6	60
Threads Per Core	1	2	4
Compute Units	4	12	16
Clock Frequency	3.0 GHz	2.66 GHz	1.05 GHz
Peak Performance	n/a	n/a	2.02 Tflops[2]
Release Date	Q2'13	Q1'10	Q4'12
Intro Price	$ 182.00	$ 999.00	$ 2,649.00

TABLE II. NVIDIA GPU DEVICE COMPARISON

Product	NVIDIA	NVIDIA	NVIDIA	NVIDIA
Model	GTX 670	m2090	K20	K40
Microarchitecture	Kepler	Fermi	Kepler	Kepler
Multiprocessors (MP)	7	16	13	15
Cuda Cores / MP	192	32	192	192
Total Cuda Cores	1344	512	2496	2880
Max Threads / MP	2048	1536	2048	2048
Total Max Threads	14336	24576	26624	30720
Clock Frequency	915 MHz	1.3 GHz	706 MHz	745 MHz
Peak Performance	n/a	1.33 Tflops[4]	3.52 Tflops[5]	4.29 Tflops[5]
Release Date	Q1'12	Q2'11	Q4'12	Q4'13
Intro Price	$ 399.00	n/a	$ 3,199.00	$ 5,499.00

The time elapsed is expressed in equation (1). In executions with multiple ranks, the rank with the longest time is taken.

$$(cyclesEnd-cyclesStart)/processorFrequency \quad (1)$$

A. Single Device Execution

To get a baseline, each device runs its corresponding ARCHER architecture simulation on its own. These results are presented in Table 3. As expected, the device with the smallest number of compute units, The Intel Core i5 CPU processor, takes the longest to compute the Monte Carlo transport CT simulation. This device is therefore taken as the base time value for which all other device speedups are computed.

As expected, the MIC architecture device with its 60 cores outperformed the 6 and 4 core CPU devices. Somewhat surprisingly, all four GPU devices including the desktop gaming GTX 670 card outperformed the MIC with the K40 GPU reaching a maximum speedup of 14.6x over the baseline Core-i5. This is largely due to the fact that the computational capability of the MIC is not being fully utilized in our current Monte Carlo implementation. As previously mentioned, each 512-bit register in the MIC can pack in and simultaneously execute sixteen single precision floating-point numbers. However, the Intel C++ compiler is not able to auto-vectorize many of the complex loop structures in the Monte Carlo transport computation. Therefore the 512-bit registers in actuality are only keeping the result of one of the sixteen calculations. Taking that into account, the Intel MIC device is posting a good result. Full utilization and implementation of the 512-bit vector units by ARCHER is still a work in progress.

B. Multiple Device Concurrent Execution

Interestingly enough, the Core-i5 CPU and the NVIDIA GTX 670 GPU pair, which is a very common desktop gaming system setup, has a faster compute time than the concurrent execution on the Intel server grade Xeon CPU and MIC coprocessor. This is due largely in part to the 9x speedup

TABLE III. SINGLE DEVICE EXECUTION TIMES AND SPEEDUPS

Device	Time	Speedup	KParticles/s
Intel Core i5-4430	1309.0	n/a	7.6
Intel Xeon X5650	875.7	1.5	11.4
Intel Xeon 5110P	211.8	6.2	47.2
NVIDIA GTX 670	153.8	8.5	65.0
NVIDIA m2090	125.7	10.4	79.6
NVIDIA K20	109.2	12.0	91.6
NVIDIA K40	89.5	14.6	111.7

achieved when adding the GTX 670 GPU to the Core-i5 CPU. Although the Intel server grade CPU tested is an older chip and the MIC implementation can be improved to obtain better vectorization of the irregular Monte Carlo algorithm, it goes to show that high performance computing is readily available in today's consumer grade heterogeneous desktop systems. Focusing on speedups, pairing an Intel MIC or an NVIDIA K40 GPU with the Intel Xeon CPU returns 5x and 10x speedups respectively. Three device concurrent execution shows a 13x speedup when running on the Intel Xeon Phi MIC, NVIDIA K40 GPU and Intel Xeon CPU. These are seen in Table 4.

TABLE IV. TWO DEVICE CONCURRENT EXECUTION TIMES

Devices			Time	Speedup
Intel Xeon X5650	Intel Xeon 5110P		172.5	7.6
Intel Core i5-4430	NVIDIA GTX 670		142.2	9.2
Intel Xeon X5650	NVIDIA m2090		117.3	11.2
Intel Xeon X5650	NVIDIA K40		85.5	15.3
Intel Xeon X5650	Intel Xeon 5110P	NVIDIA m2090	88.3	14.8
Intel Xeon X5650	Intel Xeon 5110P	NVIDIA K40	69.3	18.9

VII. CONCLUSION

In this paper we presented a new application to utilize heterogeneous computing systems concurrently. Heterogeneous computing is a new atypical and irregular platform for computing with immense computational potential that can now be fully harnessed. Previously optimized ARCHER CPU, GPU and MIC code sets have now been combined into one application for simultaneous architecture computing. The "self-service" method to device organization and operation, coupled with the ability to detect underlying device architectures in heterogeneous systems, provides balance and efficiency to concurrent execution of the irregular Monte Carlo transport algorithm on various combinations of devices. This has shown to give strong speedups as more devices are added to the concurrent execution. A 13x speedup has been shown for concurrent execution with one CPU, GPU and MIC.

REFERENCES

[1] Chrysos G., Intel Xeon Phi Coprocessor - the Architecture, Intel (2012). Found: https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner.

[2] Intel, (2013), Intel Xeon Phi Product Family Performance Rev 1.4. Retrieved July 28, 2014, http://www.intel.com/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf

[3] NVIDIA, (2009), NVIDIAs Fermi: The First Complete GPU Computing Architecture. Retrieved July 28, 2014, http://sbel.wisc.edu/Courses/ME964/Literature/whitePaperFermiGlaskowsky.pdf.

[4] NVIDIA, (2011), Tesla M-Class CPU Computing Modules: Fastest Parallel Processors For Accelerating Science. http://www.nvidia.com/docs/IO/105880/DS_Tesla-M2090_LR.pdf.

[5] NVIDIA, (2013), Tesla Kepler Family Product Overview. http://www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf.

[6] Xu X. G. et al., "ARCHER, a New Monte Carlo Software Tool for Emerging Heterogeneous Computing Environments", Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013 (SNA + MC 2013), Paris, France, October 27-31, 201.