Big Data Visualisation on the Xeon Phi

Timothy Dykes[1], Claudio Gheller[2], Marzia Rivi[3], and Mel Krokos[1]

[1] University of Portsmouth, Portsmouth, U.K.
{timothy.dykes,mel.krokos}@port.ac.uk
[2] CSCS-ETHZ, Lugano, Switzerland
cgheller@cscs.ch
[3] University of Oxford, Oxford, U.K.
rivi@physics.ox.ac.uk

Abstract. *With the increasing size and complexity of data produced by large scale astrophysical simulations, it is important to be able to exploit all available hardware in High performance Computing environments for increased throughput and efficiency. We focus on the modification and optimisation of Splotch, a scalable data visualization algorithm, to utilise the Xeon Phi, Intel's co-processor based upon the new Many Integrated Core architecture. We discuss steps taken to offload data to the co-processor and algorithmic modifications to aid faster processing on the many-core architecture and make use of the uniquely wide vector capabilities of the device, with accompanying performance results.*

Keywords: Big Data, Visualization, Xeon Phi, High Performance Computing, Astrophysics

1 Introduction

In recent years, the volume of scientific data produced by experiments, observations and numerical simulations has been increasing exponentially. This is true for many scientific domains, such as evironmental or life sciences, but in particular for astrophysics.

Some of the largest cosmological simulations, performed using sophisticated N-body codes, can describe the details of the evolution of the universe up to the present time, following the behaviour of gravitating matter represented by a hundred billion particles. Running these simulations produces output files whose size is of the order of tens of terabytes each. The fast technological progress of supercomputing systems will soon lead to simulations producing outputs with size much greater than this as we head into the exascale era.

Size does not represent the only challenge posed by scientific data. Is also essential to effectively extract all the information from the often complex data files. Software for data mining and analysis is often highly computationally demanding and essentially unusable on large datasets due to time constraints and power limitations.

Exploration and discovery through visualization represents an outstanding aid to big data processing, e.g. by providing scientists with prompt and intuitive insights enabling them to identify interesting characteristics and thus define regions of interest within which to apply further time-consuming methods. Additionally, they can be a very effective way in discovering and understanding correlations, associations and data patterns, or in identifying unexpected behaviours or even errors. However, visualization tools require high performance computing (hereafter HPC) resources, in order to overcome the issues of rendering big, complex datasets and doing so in an acceptable timeframe.

Splotch [1], our ray-casting algorithm for visualizing large-scale, particle-based datasets, addresses these issues, providing high quality graphic outputs, processing data of, ideally, any size, already efficiently exploiting a broad variety of HPC systems: multi-core processors and multi-node supercomputing systems [2], and GPUs [3]. It is essential to be able to utilise all devices that may be available within a HPC system, in order to exploit the full potential of the environment for maximum performance. Single images can be generated, or sequences of frames from different points of view to compose into an animated film of the simulation.

To this end, this paper will describe the work accomplished to enable Splotch to exploit on the new Intel PHI [4] accelerator, taking advantage of the Many Integrated Core (hereafter MIC) architecture [5], which is expected to provide, on suitable classes of algorithms, outstanding performance with power consumption being comparable to standard CPUs. We will present the MIC implementation and optimisations, performance tuning, benchmarks carried out, and the resulting performance measurements, comparing that of an OpenMP based implementation running on multiple cores of a single CPU.

2 Splotch Overview

Splotch is visualization algorithm written in pure C++ for generating 2 dimensional images from particle based datasets. The main stages of the Splotch workflow can be summarised as:

Data Load Data is read using an appropriate reader and stored in the Splotch particle structure, various readers are implemented for different datatypes.

Processing and Rasterization Data is preprocessed, performing tasks such as ranging, normalization, and applying logarithms to particle attributes, amongst others. Particles are roto-translated with reference to supplied camera and look-at positions. Active particles, those within the view frustum, are identified and assigned an RGB color value dependant on particle properties and an external color map.

Rendering For each pixel of the image, a ray is cast along the line of sight, and contributions of all encountered particles are additively accumulated. The contribution a particle may have to a particular pixel color is determined by solving the radiative transfer equation [3]

3 Splotch on the MIC

3.1 Overview of the MIC Architecture

The core ideal behind the MIC micro-architecture is obtaining a massive level of parallelism for increasing throughput performance in power restricted cluster environments. To this end Intel's flagship MIC product, the Xeon Phi, contains roughly 60 cores on a single chip, dependent on the model. The Xeon Phi acts as a coprocessor for a s tandard Intel Xeon processor. A user can execute a program natively by logging into the device itself, which hosts a Linux micro-OS, or using the device as one or more MPI processes amongst further processes running on the Xeon host. Alternatively users can offload data and portions of code to the coprocessor via series of pragma based extensions available in C++ or FORTRAN.

A brief technical description of the Xeon Phi is provided here, more detailed information can be found in the Xeon Phi whitepaper [4]. Each core has access to a 512 KB private fully coherent L2 cache, memory controllers and the PCIe client logic provide access to up to 8 GB of GDDR5 memory, and a bi-directional ring interconnect brings these components together. The cores are in-order, however up to 4 hardware threads are supported to mitigate the latencies inherent with in-order execution. The Vector Processor Unit (hereafter VPU) is worthy of note due to the utilisation of an innovative 512 bit wide SIMD capability, allowing 16 single precision (SP) or 8 double precision (DP) floating point operations per cycle, support for fused-multiply-add operations increases this to 32 SP or 16 DP floating point operations per cycle.

3.2 MIC Implementation

The rendering algorithm can be broken down into a series of phases illustrated by the execution model shown in fig. 1. While the executable runs on the Xeon host, data and processing is offloaded to the device via Intel offload clauses. A double buffered scheme has been implemented using the ability to asynchronously transfer data via a series of signal and wait clauses provided by the Intel extensions. This allows to minimise overhead due to transferring data to the device for processing, and to facilitate the rendering of datasets potentially much larger than the memory capacity available.

The first stage of rendering is a highly parallel 3D transform and colorize performed on a per-particle basis using four OpenMP threads per available core, to match the four available hardware threads, equating to roughly 240 threads. Transform parameters are precomputed, as they are the same for each particle, and computation is distributed amongst threads via an OpenMP parralel for loop. Further tuning enabled this loop to be auto-vectorized by the Intel compiler providing a large performance boost for this section.

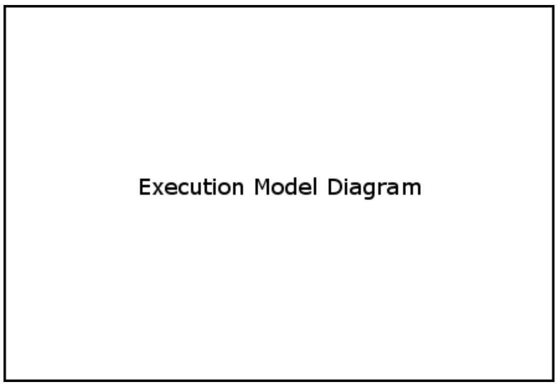

Fig. 1. Model illustrating execution pattern of the new implementation

During a pre-render stage the number of available OpenMP threads are split into groups, the number of which is dependant on a runtime parameter, allocating a buffer per group for the resulting image. The dataset is evenly distributed amongst the groups, each rendering to the assigned buffers which are finally accumulated serially to a single image.

Each group image is split into a two dimensional grid of tiles which are distributed amongst threads to avoid race conditions where multiple threads attempt to draw to a single pixel simultaneously. The number of tiles is determined through a user provided tile size parameter and the image size, and for each tile a list is generated containing the indices of all particles affecting the tile.

For the full render stage each thread processes a set number of tiles, rendering the list of particles for each tile. In this way pixels are not shared between threads and concurrent accesses are avoided. Finally when all chunks of data have been processed and accumulated, the resultant device image is copied back to the host for output.

3.3 Optimisation

Memory Usage and Data Transfer Cost of dynamic memory allocation on the Phi is relatively high [6], in order to minimise unnecessary allocations buffers are created at the beginning of the program cycle and reused throughout. Use of the MIC_USE_2MB_BUFFERS environment variable forces buffers over a particular size to be allocated with 2MB pages rather than the default 4KB, which improves data allocation rate, transfer rate and can benefit performance by potentially reducing page faults and TLB (translation look-aside buffer) misses [7]. In tests using offload clauses for compiler managed memory allocation on the device, a single process offloading to the device and reserving large buffers allocated memory roughly 2-2.5x faster having set this environment variable to 64K. The inability to asynchonously allocate offload memory means overheads incurred allocating these buffers cannot be mitigated by overlapping allocation with host activity.

Additionally, overheads in dynamic allocation and data transfer incur a penalty when running a single host process offloading to the device. In order to minimise these penalties, multiple MPI processes on the host are each allocated a subset of the device threads to exploit. In this way, the device is subdivided amongst the host MPI processes allowing for memory and data transfer to occur in parallel providing a noticeable performance increase, further details of which are given in sect. 4.1.

Vectorization The large 512 bit width SIMD capability of the MIC architecture is exploited through vectorization carried out both automatically by the compiler, and manually using Intel Initial Many-Core Instructions (IMCI) [8]. Firstly the core data structure used in Splotch rendering was re-examined, and converted from an array of structures (AoS) to a structure of arrays (SoA). The AoS method stores a large array of structures, each containing all of the information pertaining to a single particle. This aids the compiler in automatic vectorization, amongst other changes such as ensuring the correct data alignment and modifying loops to be more easily vectorized as described in Intels Vectorization guide [9] which, while providing examples for SSE, is applicable to IMCI as well. Reformatting the data storage method led to a 10% performance increase in the rototranslation and coloring phase, while the rendering phase was essentially unaffected, leading to a 3% performance increase overall.

The rendering section of the algorithm is complex and unsuitable for automatic vectorization, this is manually optimised through use of the Intel intrinsics, which map directly to IMCIs. Drawing consists of additively combining a pixels current set of RGB values with the contribution from the current particle, which is calculated by multiplying the particle color by a scalar contribution value. In order to expedite this process, up to five single precision particle RGB values (totalling 480 bits) and five contribution values are packed into two respective 512 bit vector containers. A third container contains 5 affected pixels, which are written simultaneously using a fused multiply add vector intrinsic, masked in order not to affect the final unused float value in the 16-float capable containers.

Tuning For smaller datasets where processing time is low, overheads regarding initialisation become no longer negligible. Initialisation of the device itself and OpenMP threads can cause a noticeable overhead. The impact of this can be minimised by placing an empty clause with empty OpenMP parallel section near to the beginning of the program, in order to overlap this overhead while other host activity is occurring, in this case while reading from file. Alternatively the environment variable OFFLOAD_INIT can be set to on_start to pre-initialise all available MIC devices before the program main begins execution.

Various parameters of the algorithm can be tuned to find best performance. Rendering parameters such as the number of thread groups and tile size are set to optimal defaults for the test hardware based on results of scripted tests iterating through sets of incremental potential values. These can be modified via a parameter file passed in at runtime for differing hardware.

4 Performance Analysis

Performance analysis consists of running a variety of tests on the available hardware, the Dommic facility at the Swiss National Supercomputing Centre, Lugano. Each node is based upon a dual socket eight-core Intel Xeon 2670 processor architecture running at 2.6 GHz with 32 GB of main system memory. Two Xeon Phi 5110 MIC coprocessors are available per node.

Tests are carried out using an N-Body simulation performed using Gadget [10], consisting of roughly 21 million particles; 10 million dark matter particles, 10 million baryonic matter particles and 1 million star particles. A 100 frame animation orbiting the dataset is used to measure per-frame timings.

Both the host and device code use a tile size parameter of 100 pixels, while the device code has default parameters of 8 host MPI processes, with 2 device threadgroups of 15 threads each, which has been shown in tests to be the optimal distribution for this hardware.

4.1 Results

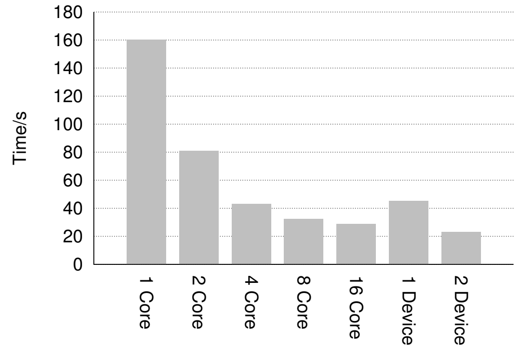

Fig. 2. Per-Frame rendering time using host with 1-16 cores vs host offloading to single and dual Xeon Phi devices

Fig. 2 shows use of a single device provides comparable results to 4 cores on the host, while two devices outperform 16 cores by 20%. The additional use of a second device provides a 2x performance improvement for the MIC algorithm. Fig. 3 shows the strongest area of improvement, the rototranslation and coloring phase, with a single device outperforming 16 cores roughly 2.5x, with roughly

3x improvement provided by using dual devices. The differing performances betweeen fig. 2 and fig. 3 are due to the rendering phase, which does not gain as significant a performance boost as the roto-translation and coloring.

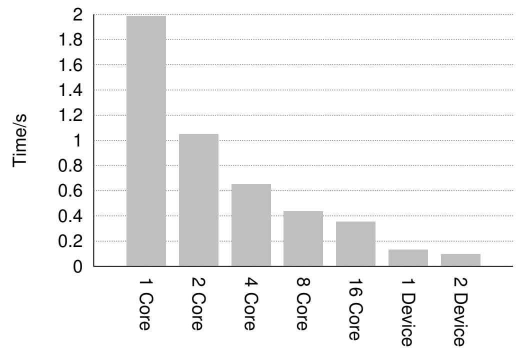

Fig. 3. Per-Frame time to rototranslate and colorize test dataset using host with 1-16 cores vs single and dual Xeon Phi devices

Fig. 4 shows comparison of per-frame render times using varying numbers of MPI processes on the host, offloading to both a single and dual devices. Subdividing the available device threads amongst host MPI processes allows to transfer data and allocate memory in parallel while also more effectively spreading the workload across the device to ensure all threads are working equally. These tests show best performance with 8 MPI processes per device, likely correspondant to the dual socket 8 core host CPU.

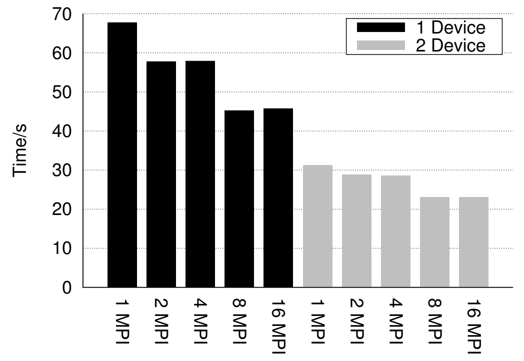

Fig. 4. Per-Frame rendering time comparing 1-16 host MPI processes (per device) offloading to 1 and 2 Xeon Phi devices

5 Conclusions

It can be seen from the results gathered so far that in some sections of code the MIC architecture excels well beyond the host, although in others a fair amount of modification is necessary to gain acceptable performance levels. The use of MPI on the host, despite implementation through an unwieldy execution script, provides a considerable performance increase especially for data heavy codes.

Further work is planned to enable the code to run on multiple host nodess utilising all available devices, with further optimisation and testing to be done in order to perfect the double buffered data transfer and processing approach to effectively visualise much larger datasets. The ability to use multiple devices across multiple nodes will allow further testing of the scalability of the approach. Features provided in the new OpenMP 4.0 specification such as the *teams* construct will enable a OpenMP based approach to threadgrouping as opposed to the manual method implemented here, if these constructs become supported by the Intel compiler in the future.

Intel released details of their second generation Xeon Phi product, codenamed Knights Landing, at the International Supercomputing Conference 2013 [11]. One important factor to note is the potential to use this as a standalone CPU rather than a co-processor. This is worthy of further investigation as it invites the dismissal of complex and time consuming data transferral methods between a host and coprocessor, while retaining the ability to run directly on the host rather than copying executables to an external device.

References

1. Splotch original reference
2. Multinode multicore reference
3. Cuda paper reference
4. Xeon Phi whitepaper reference
5. MIC architecture whitepaper reference
6. High cost of dynamic memory allocations reference
7. Benefit of 2MB buffers reference
8. Manycore instruction set reference
9. Vectorization guide reference
10. Gadget Reference
11. ISC2013 knights landing release reference