HPCC Random Access Benchmark Excels on Data Vortex™

Version 1.1* June 7 2016

Abstract

The Random Access[1] benchmark, as defined by the High Performance Computing Challenge (HPCC), tests how frequently a machine can update random elements of a table spread across global system RAM, measured in billions of updates per second (GUPS). The MPI provided implementation performs poorly on traditional InfiniBand based distributed-memory machines, as a result of updates requiring numerous small point-to-point messages between processors. In this white paper we present current performance results of the Random Access benchmark utilizing the Data Vortex (DV) network. The Data Vortex network is especially apt for this task because it can handle highly fragmented data traffic with little latency and no congestion. Benchmark tests results demonstrate that the Data Vortex network enables GUPS count to scale linearly with the number of processors, thus enabling performance into extreme scale problems and systems. An Intel Xeon based system using 64 processors and 256 cores running HPCC-MPI Random Access over InfiniBand achieves a GUPS rate of 0.048. That same system using the Data Vortex network achieves a GUPS rate of 10.0. That is over a 200 times performance increase. We describe how we designed the Data Vortex Random Access code and show performance comparisons with other systems.

Introduction

Many practical applications (such as search on graphs, linear algebra with large random sparse matrices and associated problems) rely on random memory access and large chaotic data transfers of very small packets (64-128 bits) across a network. Since current computer architectures and

* Versions reflect updates of the DV Random Access software.

[1] http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/

software usually favors transfer of large, one-dimensional buffers, the performance of these applications is strongly degraded by these operations.

MPI Random Access, is widely used by the high performance computer community to analyze random access operations performance. The Random Access benchmark is part of the High Performance Computing Challenge benchmark suite (http://icl.cs.utk.edu/hpcc). Specifically, it measures how frequently the computer can randomly transmit 64-bit packets of data and update RAM locations, measured in billions of updates per second (Giga Updates Per Second).

Conceptually the test consists on defining a vector `Table` of length `TABSIZE` that occupies half of the system memory and then writing and reading to it at `NUPDATE=4*TABSIZE` random locations. In its minimal expression, the code for a single-processor computer can be summarized with the following loop:

```
u64Int Random=1;
for (i=0; i<NUPDATE; i++) {
      Random = (Random << 1) ^ (((s64Int) Random < 0) ? POLY : 0);  // RNG
      Address = hightest_log2(TABSIZE)_bits of Random;
      Table[Address] = Table[Address] ^ Random;     // UPDATE
}
```

Figure 1: Pseudocode of conceptual GUPS code.

Then, the GUPS benchmark is the number of billions of updates per second:

$$GUPS = \frac{NUPDATES}{10^9 \times TIME}$$

The line "`(Random<<1)^(((s64Int)Random<0)?POLY:0)`" is an LFSR pseudo-random number generator based on the primitive polynomial over the Galois field of order 2, $X^{64} + X^{63}+X^{62}$, or GF(2)[2]. This random number generator (RNG) has the advantages that it is very fast to compute, its period is known (which equals 1317624576693539401 for POLY=7) and it is easy to initialize at equal spaced intervals. However, for the purpose of this test any quick RNG is appropriate: other versions of the benchmark also use congruential linear generators.

On a parallel computer there are two additional restrictions for the Random Access Benchmark:

a) The maximum look ahead buffer size for the communications of each node is 1024 addresses of 64 bits each.
b) The stream of random numbers at each node is initialized at equally spaced intervals on the interval [0..NUPDATE-1] (this is to make the output equal to the one-node program).

[2] Beker, Henry; Piper, Fred. "Cipher Systems: The Protection of Communications", Wiley (1982)

The first condition makes the test significantly more complex: the HPCC MPI code results in many more than 1000 lines of code. This complexity rises because the MPI code is usually run in systems using InfiniBand (IB) for node-to-node communication. To get optimal performance over IB, as large a data set as possible must be sent during each communication call. Therefore, when sending random update requests to a node, all updates targeted to that node should be sent in 1 message to maximize IB bandwidth. To do this, the 1024 words must be handled in complex data structures (buckets and heaps) to organize where each request is destined and to track the largest data set for a destined node. This complicates the code and introduces latency.

In order to explain the simplicity of GUPS implementation in Data Vortex Systems, we first need to explain the main features of a Data Vortex system and the Data Vortex network.

The Data Vortex System

One particular Data Vortex System, Hypatia, was used for Random Access testing over both IB and the Data Vortex network (Fig. 2). It is comprised of 32 off-the-shelf commodity servers interconnected with a Data Vortex network as well as a FDR InfiniBand network. Providing both networks on the system enables an apples-to-apples Random Access comparison of the Data Vortex network and a FDR IB network.

Each Hypatia server contains 2 Intel Xeon E5-2643v1 processors (4 core, 3.30GHz) with 128GB of memory on each processor. Connected to each processor is a separate Data Vortex - Vortex Interface Card (DV-VIC). Each DV-VIC is connected directly to the corresponding processor over an 8 lane PCIe Gen3 bus. Each DV-VIC contains sixteen 8 Gb/s SerDes (128 Gb/s total raw bandwidth) connected to the Data Vortex network. Each server also contains a 56 Gb/s FDR IB connection to a 36 port FDR IB switch.

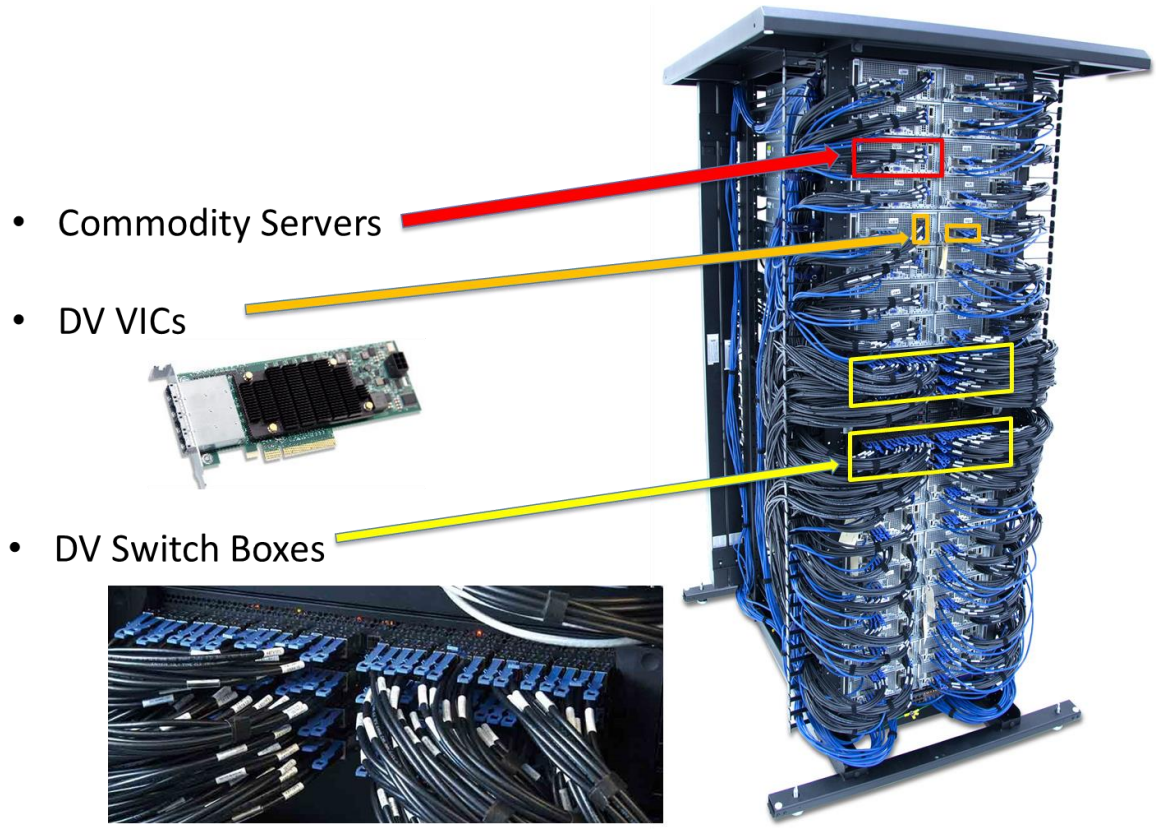

Figure 2: Data Vortex System *Hypatia* used for the comparisons in this paper.

The Data Vortex Network

A key property of the Data Vortex network is that the cost to send small fragmented messages across random nodes is the same as the cost to send long buffers to a single node. This is drastically different than competing networks.

The Data Vortex Network is comprised of high radix, low latency switches that provide fine grain (64 bit packet) congestion free switch operation. The high switch radix allows for large systems to be constructed with a low number of switching levels. The current switch chip implementation is in an Altera Stratix V FPGA. The Stratix V FPGA family contained the highest number of SerDes in an FPGA at the time of development enabling a radix 64 switch chip design. Unique features of the Data Vortex network are:

- All packets on the network are 64-bit data payload packets. Even when sending a long buffer to a single destination node, the buffer is broken into 64-bit payload data packets. Therefore, the same performance is achieved whether sending individual 64-bit payload data words to different destination nodes or sending one long buffer of data to a single node.

- Individual SerDes lanes are not ganged together to form a link as they are with most other networks. All SerDes in a Data Vortex switch chip can be connected to unique nodes. This enables high radix switch chips reducing the number of switch chip to switch chip hops for large systems. As an example, the Stratix 5 switch FPGA contains 64 SerDes which enables implementing a radix 64 Data Vortex switch within the FPGA. If 4 SerDes were required to be ganged to create a link, then the FPGA would only be able to support a radix 16 switch. Since the Data Vortex network does not require ganging, a full radix 64 switch was implemented. Therefore 2 levels of Data Vortex switch FPGAs support up to 2048 unique node addresses. If 4 SerDes ganging was required, only 128 unique node addresses could be supported using 2 levels of radix 16 switches.
- The Data Vortex switch utilizes a proprietary data flow switching mechanism that does not include an arbiter or crossbar enabling consistent performance whether the switch is lightly or heavily loaded with traffic. The application programmer does not program or train the switch to their application and there is no Software Defined Networking of a Data Vortex switch.
- The DV-VIC provides a scatter and gather mechanism to scatter data when the host processor writes a cache line of data to the Data Vortex network and gather data into cache lines when receiving data from the Data Vortex network destined to the local host processor.

Each DV-VIC in Hypatia contains sixteen 8 Gbps SerDes. Each of the VIC's SerDes are connected to a separate radix 64 Data Vortex switch chip. Therefore, there are a total of 16 Data Vortex switch chips operating in parallel where every switch chip has its own connection to every DV-VIC.

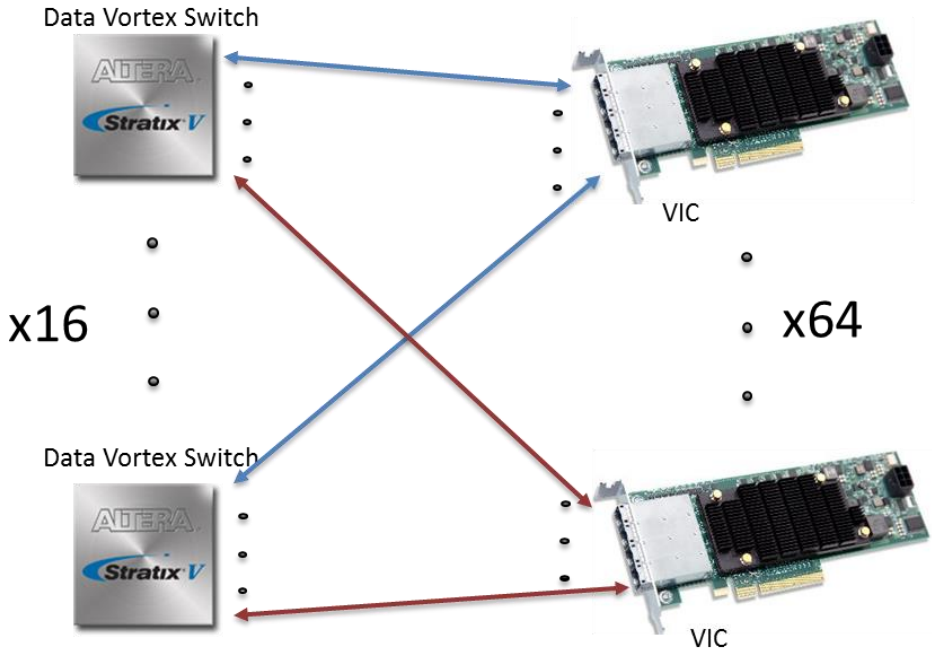

Figure 3: Hypatia's Data Vortex Network

GUPS in Data Vortex

The code for the Data Vortex system is much simpler because Data Vortex is designed to handle random communication of small 64 bit packets with little latency; thus no complicated data structures are needed. The code can instead be divided on two simple threads that execute in parallel:

Send Thread:

```
Random= CalculateInitial(Mynode);
For(i=1; i<NUM_TO_SEND_PER_NODE;i++) {
      Random = (Random << 1) ^ ( (Random<0) ? POLY : 0); // RNG
      Node = highest_log2(NODES)_bits of Random
      send Random to Node;
}
```

Receive and update thread:

```
While(still receiving) {
      receive Random;
      Address = hightest_log2(TABLESIZE)_bits of Random;
      Table[Address] = Table[Address] ^ Random; // UPDATE
}
```

Main program: launch Send and Receive threads and run until done

Figure 4: Pseudocode of the Data Vortex GUPS code. Two or more threads run simultaneously until transmission ends.

The function `CalculateInitial()` initializes the RNG at equispaced intervals across all the nodes and the period of the RNG. `NUM_TO_SEND_PER_NODE` is the total amount of packets `NUPDATE` divided by the number of nodes. The actual code has a simple buffering system that follows the 1024 words limit imposed by the rules of the benchmark. This straightforward implementation (about 300 lines of code) easily outperforms MPI by a factor larger than 50. However, we also developed more sophisticated versions that are far more powerful (and also more complex with 1526 lines of code), we show its results in the next sections.

Most of the time in the computation is spent executing three Data Vortex instructions: one that writes 64 bit words called packets to the Data Vortex network, another that read packets from the

VIC to processor memory and another that generates the headers which describe to what VIC number and VIC address each data packet is going to. We know that data has arrived to a VIC when a special counter called the *group counter* has zeroed out. The memory transfers are performed with direct memory access (DMA) so they can be overlapped with other operations. The stochastic nature of the execution of this code is illustrated in Fig. 5: the horizontal scale is time and the vertical coordinate is node number. The CPU usage is depicted by the thickness of the middle grey line.

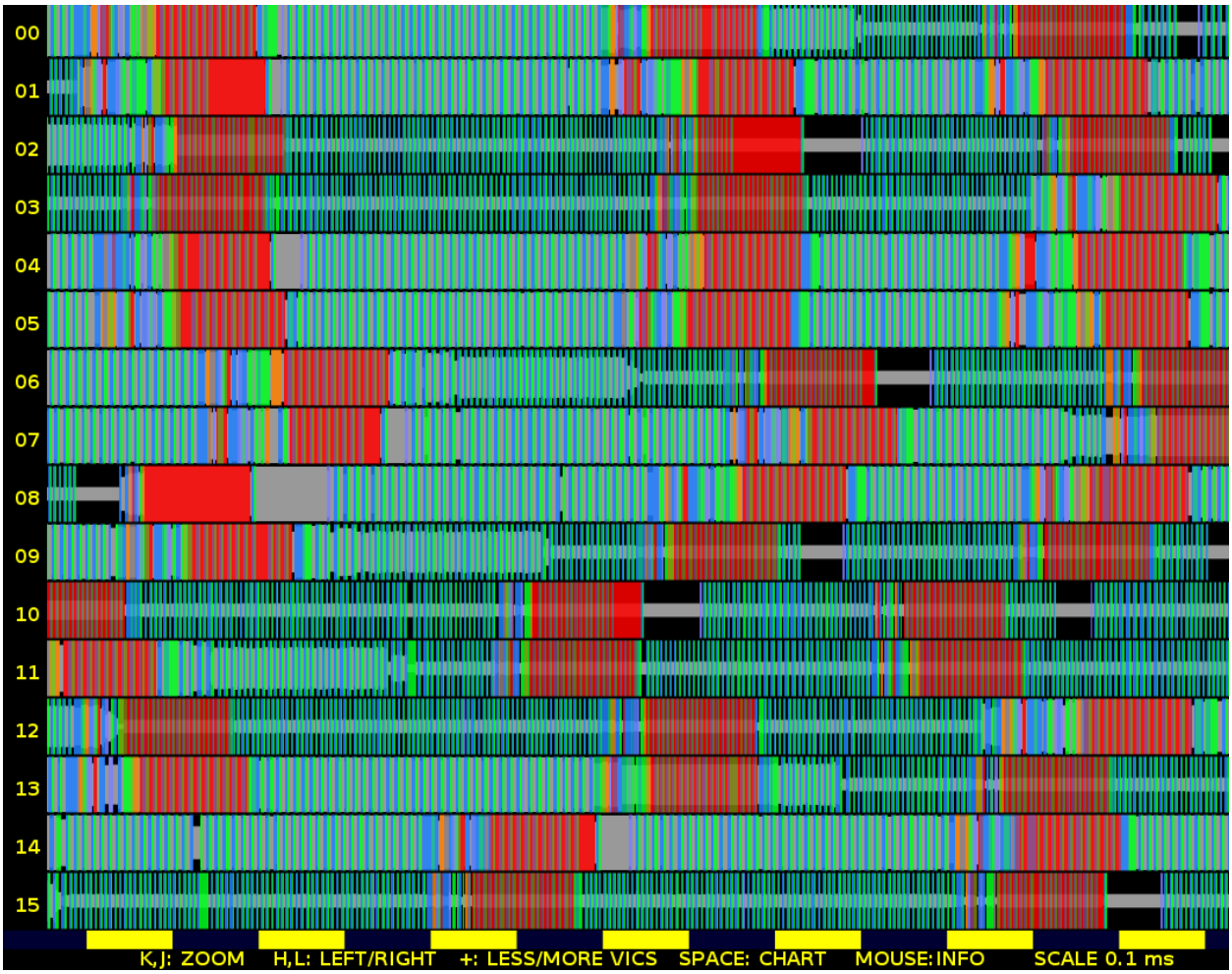

Figure 5: Graphical profile of a typical GUPS run with 16 nodes in Data Vortex, showing the stochastic nature of the running code of figure 2. The middle gray lines represent CPU utilization, and different colors represent different communication instructions. Green: wait to DMA write from processor memory to the VIC, red: DMA read wait from VIC to processor memory, blue: generate headers, violet: setting group counters, orange: VIC to processor memory kick. Data is also being transmitted in the dark regions.

Validation

The Data Vortex code was validated by comparing the output of the parallel code with the output of the serial code like that of Fig. 1. Due to the small possibility of race conditions during memory writes to the table, there is a small discrepancy of less than 0.001% of the total data, which complies abundantly with the GUPS benchmark requirement of 1%. We also validated the code in a novel way; we run the code twice and XOR both results and obtain an even lower percentage of nonzero results. The results match exactly when we use the C built in atomics __atomic_xor_fetch in the update code, but the performance drops by approximately 33%. The same result is obtained with pthread mutex locks however in this case the performance is much worse. But again, exact agreement between serial and parallel code is not required by the Random Access rules so these performance drops are not relevant.

Comparison with other systems

We performed two comparisons: one with a 32-node FDR IB network that is installed in one of our systems (Hypatia), and another with other systems that have published benchmark results.

For our local comparison we used the HPCC benchmark that is available at `icl.cs.utk.edu/hpcc`. We configured the input file so that it uses half of our system memory (128 GB per node). Then we ran our code in exactly the same machine (with same CPUs, motherboards and memory) using the Data Vortex network. The results are summarized in Fig. 6: Data Vortex is 250 to 400 times faster.

One important limitation that we discovered with GUPS is the strongly diminishing returns with respect to the number of threads running in each CPU. This is true both for Data Vortex and IB code. We suspect that this is a limitation of Intel processors; when many addresses are updated at the same time by different threads, the DRAM access rate diminishes.

In Figure 7 we compare the performance across different systems using cores as the independent variable. The reason we use cores is that most of the publically available data is in terms of cores, and not nodes.

Discussion and Conclusion

The HPCC Random Access benchmark shows that the Data Vortex can effectively communicate 64 bit packets randomly across the network at much higher speeds than other systems.

That is relevant to real world problems that send small data packets randomly across the network such as graph search operations, sparse linear algebra and the solution of partial differential equations on unstructured dynamic grids.

We believe the Intel Xeon E5-2600v1 family memory controller and/or DDR memory speeds within the commodity servers are the limiting Random Access benchmark performance factor when using the Data Vortex network on Hypatia. The Random Access benchmark was run on newer 8 node Data Vortex systems where each node (server) contains 2 Intel Xeon E5-2600v3 processors, faster DDR4 memory, and only 1 DV-VIC. These systems show an increased performance of 215 MUPs per node inferring the Data Vortex network is NOT the limiting performance factor. These results lead us to conclude the Intel Xeon processor to local DDR memory access speed is the limiting performance factor when using the Data Vortex network.

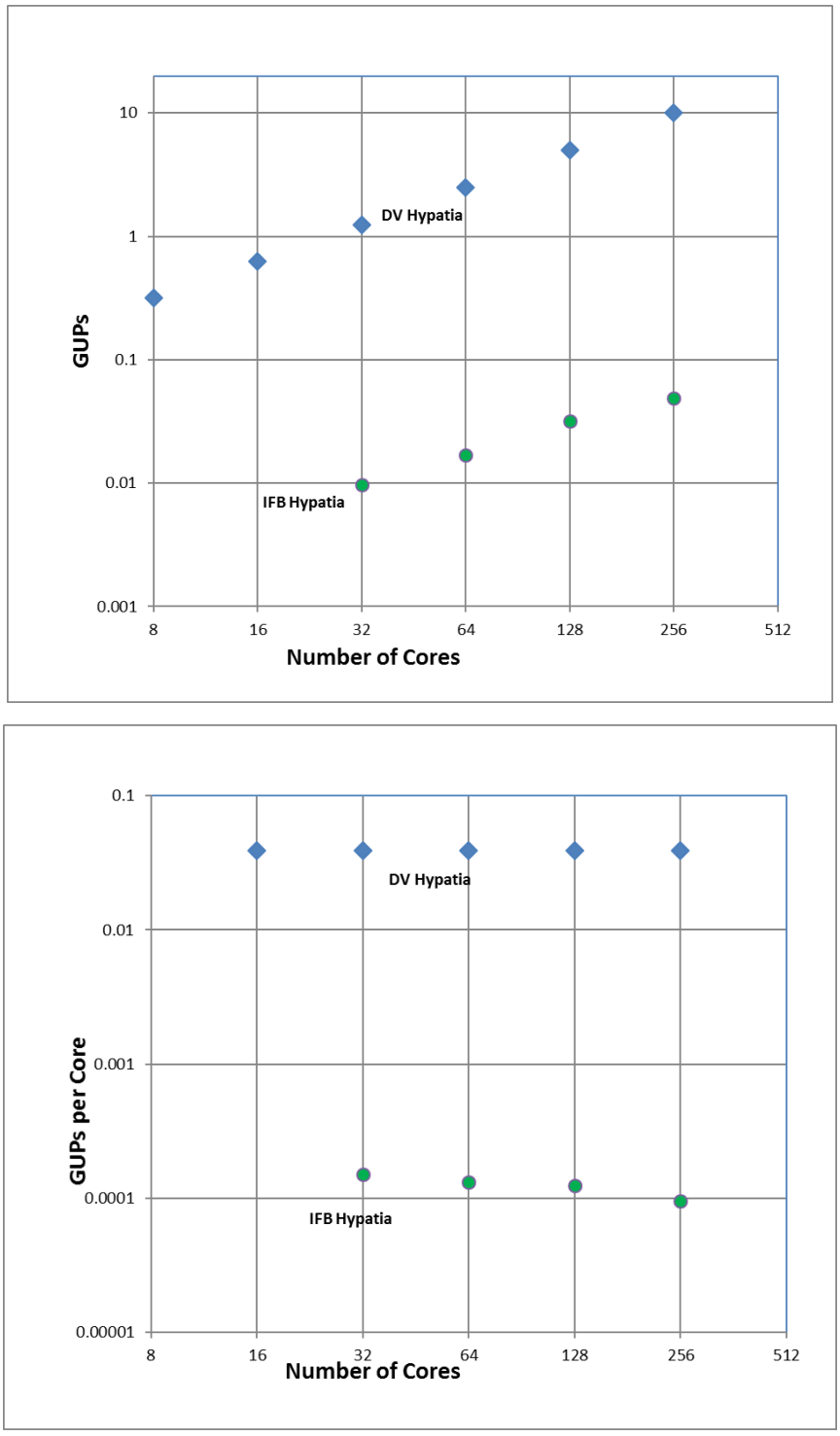

Figure 6: Random Access performance versus number of cores. The top graph depicts total system performance versus the number of cores used in the system. The lower graph depicts the average performance per core versus the number of cores used in the system.

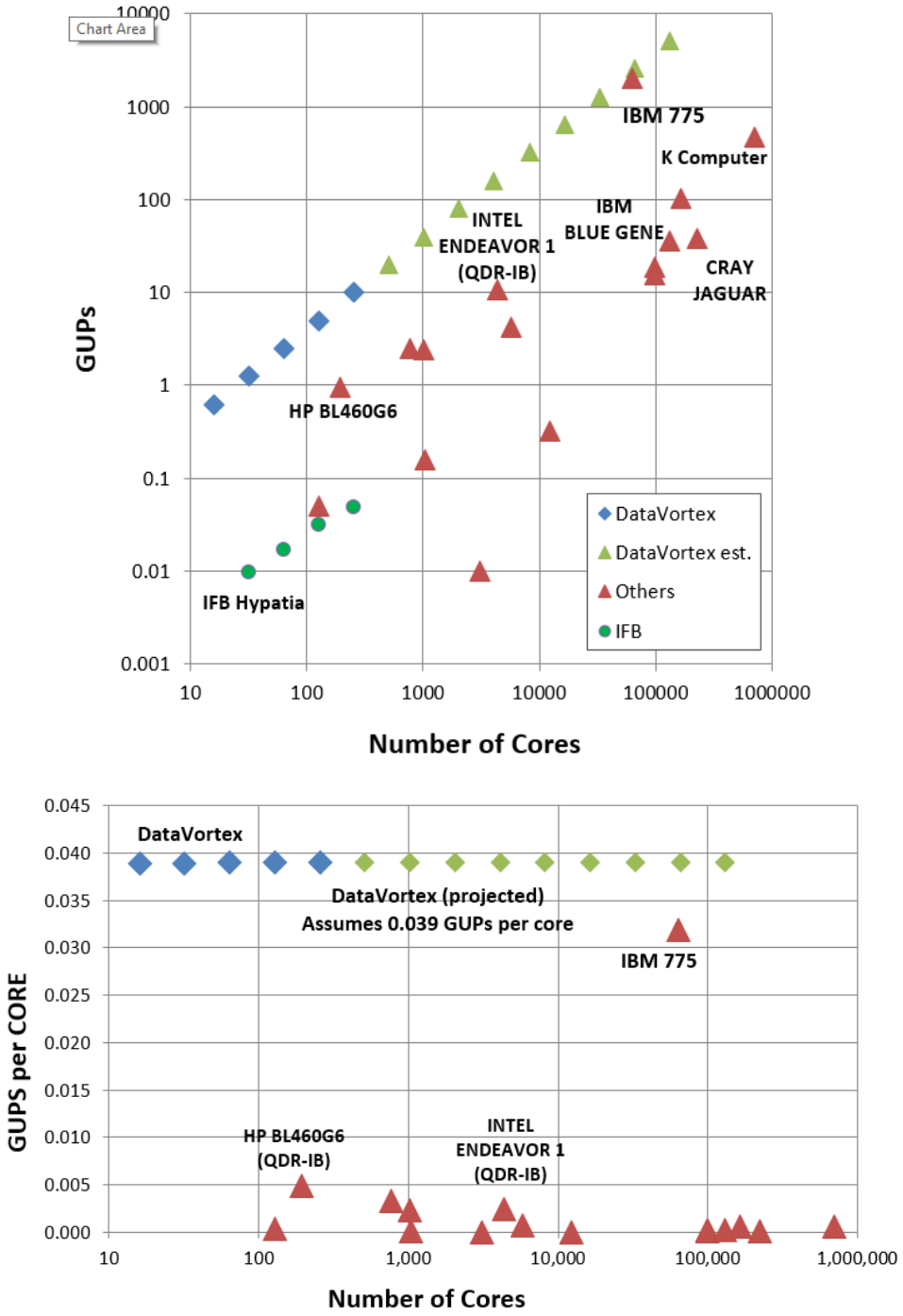

Figure 7: Performance comparison with other computers as a function of the number of cores. The top graph depicts total system performance versus the number of cores used in the system. The lower graph depicts the average performance per core versus the number of cores used in the system.