On the Performance and Energy Efficiency of the PGAS Programming Model on Multicore Architectures

Jérémie Lagravière & Johannes Langguth
Simula Research Laboratory
NO-1364 Fornebu, Norway
jeremie@simula.no
langguth@simula.no

Mohammed Sourouri
NTNU
Norwegian University of Science and Technology
NO-7491 Trondheim, Norway
mohammed.sourouri@iet.ntnu.no

Phuong H. Ha
The Arctic University of Norway
NO-9037 Tromsø, Norway
phuong.hoai.ha@uit.no

Xing Cai
Simula Research Laboratory
NO-1364 Fornebu, Norway
xingca@simula.no

Abstract*—Using large-scale multicore systems to get the maximum performance and energy efficiency with manageable programmability is a major challenge. The partitioned global address space (PGAS) programming model enhances programmability by providing a global address space over large-scale computing systems. However, so far the performance and energy efficiency of the PGAS model on multicore-based parallel architectures have not been investigated thoroughly. In this paper we use a set of selected kernels from the well-known NAS Parallel Benchmarks to evaluate the performance and energy efficiency of the UPC programming language, which is a widely used implementation of the PGAS model. In addition, the MPI and OpenMP versions of the same parallel kernels are used for comparison with their UPC counterparts. The investigated hardware platforms are based on multicore CPUs, both within a single 16-core node and across multiple nodes involving up to 1024 physical cores. On the multi-node platform we used the hardware measurement solution called High definition Energy Efficiency Monitoring tool in order to measure energy. On the single-node system we used the hybrid measurement solution to make an effort into understanding the observed performance differences, we use the Intel Performance Counter Monitor to quantify in detail the communication time, cache hit/miss ratio and memory usage. Our experiments show that UPC is competitive with OpenMP and MPI on single and multiple nodes, with respect to both the performance and energy efficiency.

I. Introduction & Motivation

The overarching complexity of parallel programming is one of the fundamental challenges that the HPC research community faces. In the last decade, the Partitioned Global Address Space model (PGAS) has been established as one possible solution for this problem. It promises improved programmability while maintaining high performance, which is the primary goal in HPC. In recent years, energy efficiency has become an additional goal. Optimizing energy efficiency without sacrificing computational performance is the key challenge of energy-aware HPC, and an absolute requirement for attaining Exascale computing in the future.

In this study we investigate whether PGAS can meet the goals of performance and energy efficiency. We focus on UPC, one of the most widely used PGAS implementations, and compare it to MPI and OpenMP. OpenMP offers ease of programming for shared memory machines, while MPI offers high performance on distributed memory supercomputers.

PGAS combines these advantages through a simple and unified memory model. On a supercomputer, this means that the programmer can access the entire memory space as if it is a single memory space that encompasses all the nodes. Through a set of functions that makes data *private* or *shared*, PGAS languages ensure data consistency across the different memory regions. When necessary, shared data is transferred automatically between the nodes through a communication library such as GASnet [1].

Recent studies [2], [3] advocate the use of PGAS as a promising solution for HPC. Many have focused on the evaluation of PGAS performance and UPC in particular [4]–[10]. However, the previous UPC studies have not taken energy efficiency into consideration. This motivates us to investigate UPC's energy efficiency and performance using the latest CPU architecture with advanced support for energy and performance profiling.

For our evaluation we use the well-established NAS Benchmark. We use MPI, OpenMP, and UPC implementations [11], [12] to compare the performance and energy efficiency of the different programming models. The energy measurements of the single-node system are obtained by using Intel PCM [13]. The multi-node performance measurements are obtained on an Intel Xeon based supercomputer , the energy measurement are obtained on this platform by using High Definition Energy Efficiency Monitoring (HDEEM) [29]. We provide an analysis of the single-node measurements in order to explain the difference in performance and energy efficiency, by focusing on the cache performance and the memory traffic of MPI, OpenMP and UPC.

This paper improves upon previous works [4]–[6], [8]–[10] by: (1) providing measurements for a larger number of nodes and cores for MPI, OpenMP and UPC on recent single-node and multi-node systems (up to 1024 physical cores); (2) including energy measurements obtained on both a single-node system and a multi-node system; (3) making an effort to understand the differences in energy efficiency and performance between UPC, OpenMP and MPI.

The remainder of this paper is organized as follows: Section II briefly presents the UPC framework and why we have chosen this programming language. Section III describes the benchmark chosen for this study. Section IV explains the hardware and software set-up used for running our experiments, the results of which are presented in Section V and discussed in Section VI. Section VII concludes the paper.

II. PGAS Paradigm and UPC

PGAS is a parallel programming model that has a logically partitioned global memory address space, where a portion of it is local to each process or thread. A special feature of PGAS is that the portions of the shared memory space may have an affinity for a particular process, thereby exploiting locality of reference [14], [15].

In the PGAS model, each node has access to both private and shared memory. Accessing the shared memory to either read or write data can imply inter-node communication which is handled automatically by the runtime. Remote access to memory work in an RDMA (Remote Direct Memory Access) fashion, using one-sided communication. However, most PGAS languages are built over a low-level communication layer which limits their physical capabilities. Thus, RDMA is available only if the underlying hardware and software support it.

In recent years several languages implementing the PGAS model have been proposed. UPC, which is essentially an extension of the C language, was one of the first ones and also one of the most stable [16]. Other members of the PGAS family of languages include the Fortran counterpart, Coarray Fortran [17], X10 [18], and Cray Chapel [19]. In addition, libraries such as Global Arrays [20] and SHMEM/OpenSHMEM [21] which implement PGAS functionality are available.

The key characteristics of UPC are: a parallel execution model of Single Program Multiple Data (SPMD); distributed data structures with a global addressing scheme, with static or dynamic allocation; operators on these structures, with affinity control; and copy operators between private, local shared, and distant shared memories.

Additionally, multiple open-source implementations of the UPC compiler and runtime environment are available, in particular Berkeley UPC [22], GCC/UPC [23] and CLANG/UPC [24].

III. The NAS Benchmark

The NAS Benchmark [25] consists of a set of kernels that each provides a different way of testing the capabilities of a supercomputer. The NAS Benchmark was originally implemented in Fortran and C. We use both the Fortran and C implementations for OpenMP and MPI, as well as the UPC version of the benchmark [12]. For our study, we select four kernels: Integer Sort (IS), Conjugate Gradient (CG), Multi-Grid (MG), and Fourier Transformation (FT).

CG refers to the *conjugate gradient* method used to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix.

TABLE I
Experimental setup: hardware

	Single Node	Multi Node
CPUs	2	2 (per node)
Cores	16	24 (per node)
CPU model	Intel Xeon E5-2650 @ 2.00GHz	Intel Xeon E5-2680 v3 @ 2.50GHz
Interconnect	N/A	Infiniband: 6.8 GB/s

MG is a simplified *multigrid* kernel. Multigrid (MG) methods in numerical analysis solve differential equations using a hierarchy of discretizations.

FT is a three-dimensional partial differential equation solver using *Fast Fourier Transformations*. This kernel performs the essence of many spectral codes. It is a rigorous test of all-to-all communication performance.

IS represents a large *integer sort*. This kernel performs a sorting operation that is important in particle method codes. It evaluates both integer computation speed and communication performance.

CG, IS, MG and FT are selected because they are the most relevant ones: stressing memory, communication and computation. They involve very different communication patterns, which is important for evaluating the performance of the selected languages (see Section V). The other kernels in the NAS Benchmark are of limited relevance to this study. See [25] for their descriptions.

IV. Experimental Setup

In this section we describe the software and hardware solutions that we used to carry out our experiments. We ran the NAS kernels both on a single-node machine and on Taurus [26], a supercomputer operated by Dresden University of Technology, using a varying number of cores and nodes.

A. Hardware

Table I shows the specifications of the systems used in our experiments. The single-node system is equipped with Intel Sandy Bridge processors and the multi-node supercomputer is equipped with Intel Haswell processors.

B. Software

On the single-node machine we used UPC version 2.22.0, the Intel Compiler version 15.0.1, and MPI Library 5.0 Update 2 for Linux. On the multi-node supercomputer we used UPC version 2.22.0 and using Bull XMPI version 1.2.8.4.

1) UPC: On the multi-node supercomputer, the UPC compiler and runtime were built with a specific option enable-segment-large in order to support large memory systems. The UPC applications were compiled using a fix number of threads and a fix network choice: *upcc -T1024 -network=mxm.* The UPC implementations were run using a fixed number of threads and fixed shared heap size and thread binding: *upcrun -n number_of_processes -bind-threads -shared-heap=3765MB ./application.* On the single-node system the UPC application were using the symmetric multiprocessing (smp) network conduit. The following environment variables were defined for *UPC GASNET_PHYSMEM_MAX=63G* which indicates to GASNET [1] to use 63GB of RAM on each node. We also used the environment variable *GASNET_PHYSMEM_NOPROBE=1* which indicates to GASNET to avoid memory detection on each node. Except for FT-D-16 where we used a modified GASNET environment variable: *GASNET_PHYSMEM_MAX=254G* because FT requires more RAM than the other kernels.

2) MPI: On the multi-node supercomputer and single-node system MPI applications were compiled using the *-O3 -mcmodel=medium* flag in order to handle larger data in memory.

3) OpenMP: On the single-node system we used OpenMP version 4.0 and we used *numactl* and the *OMP_NUM_THREADS* environment variable to bind the threads and define the number of threads.

4) Benchmarking: For each measurement we executed three separate runs and reported the best result. Doing so filters out the OS interference.

On the single-node machine we used size Class **C** [11], [27] for each kernel. For CG, Class C, the number of rows is 150000. For MG, Class C, the grid size is $512 \times 512 \times 512$. For FT, Class C, the grid size is $512 \times 512 \times 512$. For IS, Class C, the number of keys is 2^{27}. On the multi-node supercomputer we used for each kernel Class **D**, except for IS which is not available in Class **D** in the UPC implementation [12]. For CG, Class D, the number of rows is 1500000. For MG, Class D, the grid size is $1024 \times 1024 \times 1024$. For FT, Class D, the grid size is $2048 \times 1024 \times 1024$.

In our study the comparison between the different implementations is fair as the number of operations (expressed in Million Of Operation - MOP) is identical in all implementations (OpenMP / MPI / UPC). The number of MOP is reported by the benchmarks. For example, the number of MOP for CG in size C is precisely 143300 for all implementations (OpenMP, MPI and UPC).

Each kernel was run using up to 1024 CPU cores and thus 64 nodes of the supercomputer. However, for CG, limitations in the UPC implementation prevent us from using more than 256 cores, see Figure 1.

Sizes **C** and **D** provide data sets that are sufficiently large to exceed the cache size of the test systems [28] [27].

5) Thread Binding: Thread binding or thread pinning is an approach that associates each thread with a specific processing element. In our experiments we applied thread/process binding to the physical cores.

C. Energy Measurements

1) Multi-Node Platform: On the multinode platform we have chosen High Definition Energy Efficiency Monitoring (HDEEM) [29]. HDEEM is a hardware based solution for energy measurements, meaning that additional hardware is used to measure energy in a supercomputer. HDEEM is an intra-node measurement tool, which indicates that the additional hardware that performs the energy measurements is located inside each node of the supercomputer [30]. The authors in [29] define four criteria, including spatial and temporal granularity, accuracy and scalability, for power and energy measurement. HDEEM can achieve an accuracy of 99.5% over 270 nodes by using an appropriate filtering approach to prevent the aliasing effect. HDEEM is based on an FPGA solution to achieve spatial fine-granularity by measuring every blade, CPU and DRAM power separately, with a sampling rate of 1,000 Sa/s over 500 nodes [31]. Our measurements do not take into account the energy consumption of the network between the nodes of the multi-nodes platform.

2) Single-Node Platform: On the single-node platform we have chosen a software based solution in order to measure the CPU and RAM energy consumption and memory and cache usage. Intel Performance Monitor (Intel PCM) is used for the energy efficiency experiments on the single-node platform and to measure memory and cache usage [13]. Intel PCM uses the Machine Specific Registers (MSR) and RAPL counters to disclose the energy consumption details of an application [32]. Intel PCM is able to identify the energy consumption of the CPU(s) and the RAM. Quick-Path Interconnect energy consumption is not taken into account because Intel PCM was unable to provide measurement on the chosen hardware platform.

The RAPL values do not result from physical measurement. They are based on a modeling approach that uses a "set of architectural events from each core, the processor graphics, and I/O, and combines them with energy weights to predict the package's active power consumption" [33]. Previous studies have demonstrated that using a counter-based model is reasonably accurate [34]–[37]. The RAPL interface returns energy data. There is no timestamp attached to the individual updates of the RAPL registers, and no assumptions besides the average update interval can be made regarding this timing. Therefore, no deduction of the power consumption is possible other than averaging over a fairly large number of updates. For example, averaging over only 10 ms would result in an unacceptable inaccuracy of at least 10% due to the fact that either 9, 10, or 11 updates may have occurred during this short time period [38]. In our experiments the execution time of each kernel is always above a second, thus estimating the power data by averaging reported Joules over time is accurate. For convenience, in our study, we use the word "measurement" to mention the values reported by Intel PCM.

We do not track and report the energy consumption curve along each kernel's execution time line, instead we choose to report the total energy consumption ($Joules$) and the average power consumption ($Watt = Joules/seconds$).

V. Results

In this paper, we consider two metrics to measure the performance and energy efficiency. To evaluate the performance we use Million Operations Per Second (MOPS). This metric is used for both the multi-node measurements and single-node measurements. To evaluate the energy efficiency, Millions Operations Per Seconds over Watts (MOPS per Watt) is used as the energy efficiency metric. This metric is used for both the multi-node measurements and single-node measurements. The *500 Green - Energy Efficient High Performance*

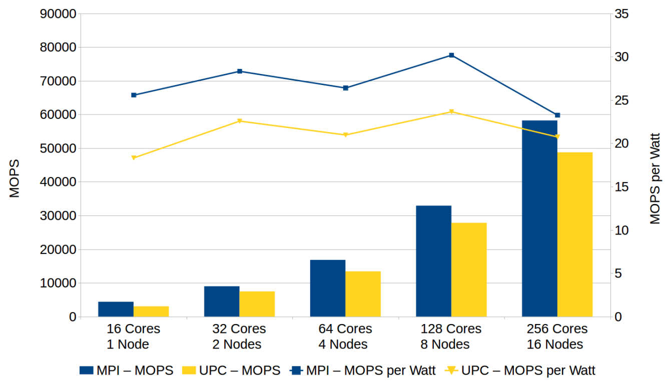

Fig. 1. Multi-node performance and energy efficiency of the CG kernel - Class D

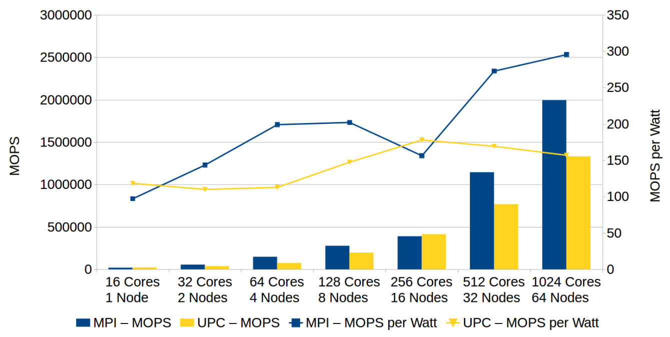

Fig. 2. Multi-node performance and energy efficiency of the MG kernel - Class D

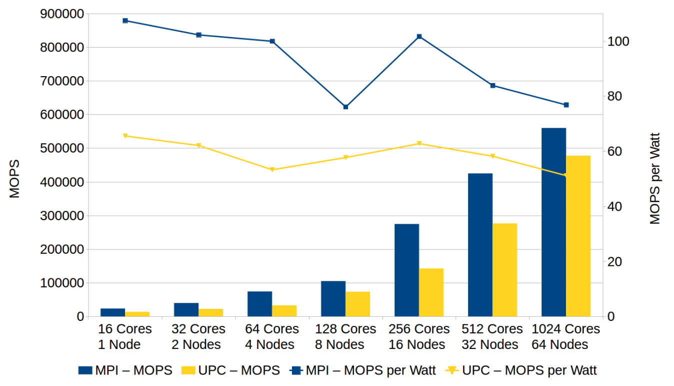

Fig. 3. Multi-node performance and energy efficiency of the FT kernel - Class D

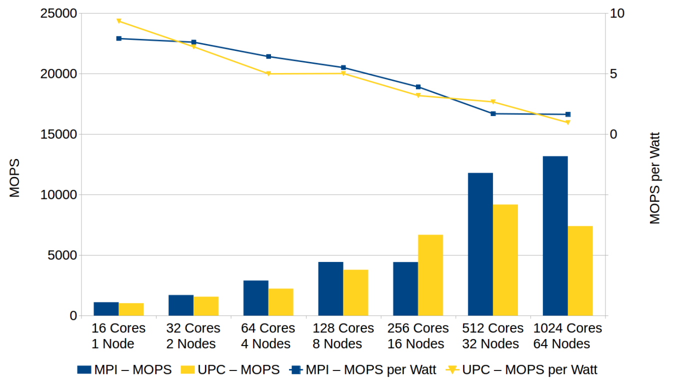

Fig. 4. Multi-node performance and energy efficiency of the IS kernel - Class C. On the left Y-axis the scale has been shifted for readability purposes.

Computing Power Measurement Methodology [39] advises this measurement methodology. We remark that MOPS per Watt is equivalent to MOP/Joules: $(MOP/seconds)/Watt = (MOP/seconds)/(Joules/seconds) = MOP/Joules$

For single-node measurements, we use the following notation: [kernel name]-[number of threads/processes]-[1S / 2S]. For instance, *CG-8-2S* stands for ”Conjugate Gradient kernel running on 8 threads (or processes for MPI) spread over two sockets”.

For multi-node measurements, we use the following notation [kernel name]-[class]-[number of threads/processes]. For instance, MG-D-256 stands for the ”Multigrid kernel of class D running on 256 threads (or processes for MPI)”. Each node always used the maximum number of physical cores, *i.e.* 16.

A. Measurement on Multi-Node Architecture

To the best of our knowledge, this is the first investigation of UPC’s energy efficiency. Our experimental results show that the energy efficiency of UPC, MPI, and OpenMP implementations scale over the number of cores and are comparable to each other. Figures 1-4 show the multi-node performance expressed in MOPS and the energy efficiency expressed in MOPS per Watt, for the four kernels implemented in UPC and MPI. Each kernel ran on up to 1024 cores, except CG where the UPC implementation cannot run on more than 256 threads.

The performance results (bars) show that the CG, MG and FT kernels scale over the number of cores independently of the language. MPI is a clear winner when running on more than 32 cores, however UPC achieves a performance that is close to that of MPI, particularly for the CG and MG kernels. These results match previous studies, in particular [9].

IS is aside in terms of performance, because of the size of the data to process, size C, is smaller than size D and causes both UPC and MPI not being able to scale on more than 256 cores. For 512 and 1024 cores runs, IS-C does not deliver good performance as the communication cost outbalances the computation performance.

Figures 1-4 show a complex relation between performance (bars) and energy efficiency (lines). While the performance of both UPC and MPI go up with increasing numbers of cores and nodes, the energy efficiency is at best staying constant or diminishing. The only exception is the MG benchmark, where higher energy efficiency is achieved by using more cores and nodes. For the IS benchmark that uses the size of Class C, in particular, the energy measurements obtained by HDEEM are gradually dominated by the non-scalable initialization phase as the number cores increases. (In comparison, the energy measurements obtained by Intel PCM on the single-node system do not include the initialization phase.)

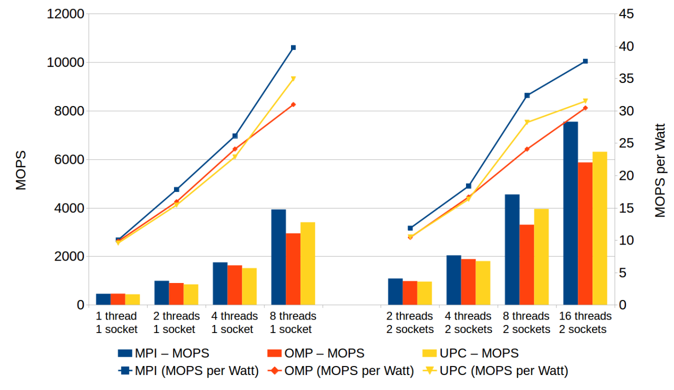

Fig. 5. Single-node performance and energy efficiency of the CG kernel - Class C

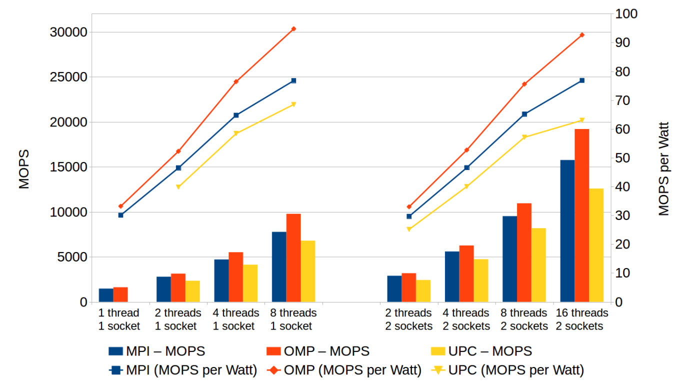

Fig. 7. Single-node performance and energy efficiency of the FT kernel - Class C

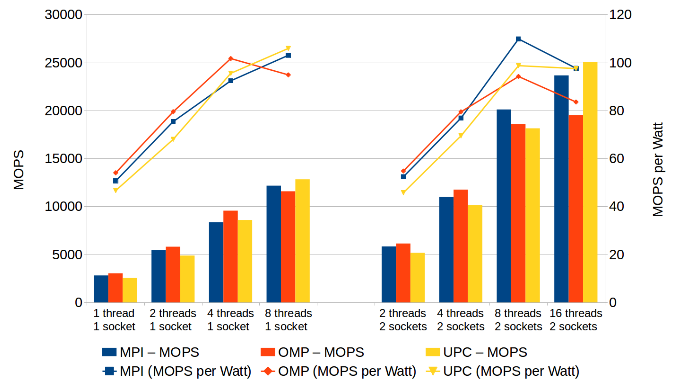

Fig. 6. Single-node performance and energy efficiency of the MG kernel - Class C

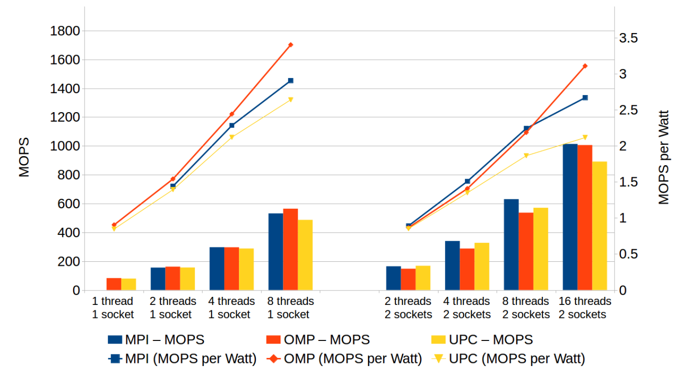

Fig. 8. Single-node performance and energy efficiency of the IS kernel - Class C

B. Measurements on Single-Node Architecture

Our experimental results show that the energy efficiency of UPC, MPI, and OpenMP implementations scale over the number of cores and are comparable to each other. In this section we give more details about these results and in Section VI we provide an analysis of the difference in performance between UPC, OpenMP, and MPI.

Figures 5-8 show both the performance expressed in MOPS for the four kernels (bars) and the energy efficiency expressed in MOPS per Watt (lines). Each kernel ran on up to 16 cores. As the single-node system is equipped with two CPUs we ran the kernels on both one socket and two sockets for 2,4 and 8 threads counts. Each of these figure shows the results for all three programming models. The energy efficiency results show that the kernels scale over the number of threads/cores independently of the language. In Figures 5-8, when measured on one-socket, the energy consumption of the idle socket and memory controller were not taken into account. There is no clear winner since none of the chosen languages is better than the other two competitors for all the kernels.

UPC was not able to run FT-C on one thread and MPI was not able run IS-C on one thread.

Even though there is no global *winner* in the obtained single-node measurements, UPC is able to compete with both OpenMP and MPI. UPC arrives in second place for CG-8-1S, CG-8-2S, CG-16-2S, IS-2-2S, IS-4-2S, IS-8-2S and MG-4-1S.

As described in [9], on a single-node platform, UPC scales well over more CPU cores and competes well with OpenMP and MPI. However, the performance of MPI or OpenMP is better in many cases.

For general-purpose architectures with high static power such as Intel Sandy Bridge, energy efficiency is directly connected to performance results. Therefore, the best results in energy efficiency are achieved, in most cases, for the kernels and thread-counts delivering the highest performance in MOPS.

However, by looking closely at the performance and performance per watt, it is possible to highlight that running a program in MPI, OpenMP or UPC over only 1 socket instead of two, when it is possible, delivers better energy efficiency. In Table II, we report values for CG in order to show a comparison between runs over 2,4 and 8 cores using 1 or 2 sockets. Table II is divided in two parts: the first part where the measurements on 1 socket do not include the measurement of the second idling socket, the second part where the idle socket measurements are included in the reported values. Table II corresponds to Figure 5, we chose to only represent the value of CG because the results of CG in terms of energy

TABLE II

COMPARISON OF COMPUTATION AND ENERGY PERFORMANCE OF CG OVER 2,4, AND 8 THREADS USING 1 AND 2 SOCKETS.

SYNTAX: *CG-2-2S / CG-2-1S* STANDS FOR "MEASUREMENTS FROM CG RUNNING OVER 2 THREADS USING 2 SOCKETS COMPARED TO MEASUREMENTS FROM CG RUNNING OVER 2 THREADS USING 1 SOCKET"

Second socket not included when not used	MPI			OpenMP			UPC		
CG	Performance gain in %	Additional energy usage in %	Change in MOPS / Watt in %	Performance gain in %	Additional energy usage in %	Change in MOPS / Watt in %	Performance gain in %	Additional energy usage in %	Change in MOPS / Watt in %
CG-2-2S / CG-2-1S	9.64	50.36	-33.49	8.92	52.85	-34.57	13.50	47.20	-32.06
CG-4-2S / CG-4-1S	16.53	42.22	-29.69	15.84	44.42	-30.76	19.18	40.03	-28.59
CG-8-2S / CG-8-1S	15.71	22.83	-18.58	12.07	28.63	-22.26	16.14	23.85	-19.24

Second socket always included even when not used	MPI			OpenMP			UPC		
CG	Performance gain in %	Additional energy usage in %	Change in MOPS / Watt in %	Performance gain in %	Additional energy usage in %	Change in MOPS / Watt in %	Performance gain in %	Additional energy usage in %	Change in MOPS / Watt in %
CG-2-2S / CG-2-1S	9.64	8.86	-8.14	8.92	10.02	-9.10	13.50	5.28	-5.01
CG-4-2S / CG-4-1S	16.53	7.83	-7.26	15.84	8.68	-8.00	19.18	4.79	-4.58
CG-8-2S / CG-8-1S	15.71	0.63	-0.62	12.07	3.55	-3.43	16.14	0.23	-0.21

efficiency difference between one socket and two sockets are representative of all the kernels. In the first part of Table II, we can see that CG-2-2S in MPI (CG running over two threads using two sockets) delivers better performance than CG-2-1S in MPI (CG running over two threads using one socket): +9.64% MOPS, but it costs +50,36% in energy (Joules) and delivers -33.49% MOPS per Watt. In the second part of Table II, we can see that the same comparison, including values from the second idle socket, gives a similar conclusion as before: using two sockets consumes +8.86% energy and delivers -8.14% in MOPS per Watt. These observations are also visible in Figure 5, 6, 7 and 8: the lines representing the MOPS per Watt are higher for kernels using one socket for equivalent threads/process count than that of kernels using two sockets.

VI. DISCUSSION

In this section, we give an analysis of the differences in performance and energy efficiency that were observed in the previous section.

Intel PCM provides access to metrics such as memory traffic and hit and miss rates for L2 and L3 cache. In this section we will use measurements of these metrics to analyze the differences in performance and energy efficiency among OpenMP, MPI, and UPC. Table III shows the measurements obtained via Intel PCM. In Table III, we use different metrics: Memory traffic ($read + write$) expressed in GigaBytes, L2 and L3 cache hit ratio, and L3 access given in millions of accesses. The results are given for each kernel (CG, MG, FT and IS) over 1, 2, 4 and 8 cores using one socket and 2, 4, 8 and 16 cores using two sockets.

By using Table III, we can for instance analyze the performance and energy efficiency of UPC, OpenMP and MPI for CG-4-2S and CG-8-2S presented in Figure 5. In CG-8-2S, MPI obtains the best performance and energy efficiency, UPC is second in performance and energy efficiency and OpenMP is third. By using the data from Table III it is possible to explain this result: UPC has an increased L2 cache hit ratio compared to OpenMP ($0.56 > 0.14$). MPI is better than OpenMP in CG-8-2S, because it has fewer L3 accesses ($11211 < 17023$) and better L2 hit ratio ($0.57 > 0.14$).

In MG-16-2S, UPC obtains the best performance compared to MPI and OpenMP because it has lower memory traffic than OpenMP ($452.36\ GB < 520.69\ GB$) and MPI ($452.36\ GB < 487.4\ GB$), better L3 cache hit ratio than OpenMP ($0.28 > 0.19$) and MPI ($0.28 > 0.2$) and fewer L3 accesses than OpenMP ($2782 < 3663$) and MPI ($2782 < 4390$).

In FT-16-2S, OpenMP obtains the best performance and energy efficiency due to having lower memory traffic than UPC ($497.33\ GB < 1009\ GB$) and MPI ($497.33\ GB < 878.8\ GB$), better L3 hit ratio than UPC ($0.76 > 0.26$) and MPI ($0.76 > 0.43$) and a lower volume of L3 accesses than UPC ($1681 < 22562$) and MPI ($1681 < 8319$).

In IS-8-2S, MPI is better than OpenMP and UPC because it has higher L3 cache hit ratio than OpenMP ($0.13 > 0.04$) and UPC ($0.13 > 0.07$) and a lower level of L3 accesses than OpenMP ($2815 < 4550$) and UPC ($2815 < 5143$). UPC in IS-8-2S wins over OpenMP because of its slightly better L3 cache hit ratio ($0.07 > 0.04$).

Globally the results presented show a correlation between the number of cores used and the achieved power efficiency. However we noticed in Table II a possible trade-off between performance and performance per watt by running programs over 1 or 2 sockets depending on what is the chosen goal (pure performance or energy efficiency). We used Table III to analyze the difference in performance between MPI, OpenMP and UPC over various threads/processes counts and sockets counts. We saw that by considering the memory traffic, L3 cache hit ratio, L2 cache hit ratio and the volume of access to L3 cache, that more insight into the performance can be obtained.

VII. CONCLUSION

In this study, we have investigated and provided insights into UPC energy efficiency and performance using the latest CPU architecture with advanced support for energy and performance profiling. We have measured the energy efficiency and

TABLE III
MEMORY TRAFFIC, L3 CACHE HIT RATION AND L2 CACHE HIT RATIO, FOR CG, MG, FT AND IS IMPLEMENTED IN MPI, OPENMP AND UPC
RESULTS ARE PRESENTED FOR 2,4 AND 8 CORES USING ONE AND TWO SOCKETS AND 16 CORES USING TWO SOCKETS

	MPI				OpenMP				UPC			
CG - Size C	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio
1 Threads - 1 socket	833.52	0.81	16049	0.14	834.71	0.81	16049	0.14	834.58	0.81	16049	0.14
2 Threads - 1 socket	846.14	0.77	16883	0.27	836.03	0.81	16049	0.14	856.01	0.77	16883	0.27
2 Threads - 2 sockets	845.22	0.77	17391	0.27	841.93	0.81	16798	0.14	854.4	0.77	17725	0.27
4 Threads - 1 socket	854.25	0.77	16883	0.27	834.26	0.81	16049	0.14	878.64	0.77	16883	0.27
4 Threads - 2 sockets	859.01	0.77	17474	0.27	842.96	0.81	16823	0.14	881.10	0.77	17817	0.27
8 Threads - 1 socket	878.89	0.74	11345	0.56	832.17	0.81	16049	0.14	909.09	0.73	12042	0.56
8 Threads - 2 sockets	881.81	0.74	11211	0.57	841.31	0.80	17023	0.14	912.42	0.73	11827	0.56
16 Threads - 2 sockets	908.34	0.74	11535	0.565	840.23	0.80	17014	0.14	968.8	0.72	12439	0.56

	MPI				OpenMP				UPC			
MG - Size C	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio
1 Threads - 1 socket	388.53	0.38	1174	0.83	372.25	0.31	1103	0.87	369.09	0.52	810	0.78
2 Threads - 1 socket	459.01	0.35	1540	0.82	437.43	0.28	1439	0.85	435.71	0.48	1100	0.75
2 Threads - 2 sockets	383.96	0.42	1005	0.83	370.92	0.31	1084	0.87	370.87	0.53	794	0.78
4 Threads - 1 socket	545.86	0.26	3031	0.76	513.15	0.23	2078	0.84	436.42	0.46	1178	0.75
4 Threads - 2 sockets	406.04	0.39	1275	0.81	434.59	0.27	1448	0.86	373.88	0.52	850	0.77
8 Threads - 1 socket	476.24	0.20	4355	0.33	521.27	0.20	3415	0.79	453.60	0.28	2843	0.49
8 Threads - 2 sockets	400.37	0.23	2722	0.38	510.78	0.23	2061	0.84	388.32	0.33	1861	0.57
16 Threads - 2 sockets	487.40	0.20	4390	0.32	520.69	0.19	3663	0.795	452.36	0.28	2782	0.49

	MPI				OpenMP				UPC			
FT - Size C	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio
1 Threads - 1 socket	617.90	0.38	8905	0.56	471.60	0.79	1400	0.43				
2 Threads - 1 socket	852.26	0.50	6244	0.63	470.57	0.79	1437	0.42	958.65	0.31	16610	0.56
2 Threads - 2 sockets	853.49	0.51	6063	0.63	471.54	0.80	1378	0.41	959.68	0.31	16587	0.57
4 Threads - 1 socket	867.25	0.49	6753	0.62	475.96	0.79	1446	0.41	960.74	0.31	16597	0.59
4 Threads - 2 sockets	865.76	0.50	6291	0.64	470.93	0.79	1424	0.41	960.21	0.31	16432	0.58
8 Threads - 1 socket	881.36	0.42	8767	0.64	491.11	0.76	1601	0.43	977.94	0.28	19582	0.56
8 Threads - 2 sockets	873.64	0.48	6941	0.64	475.76	0.78	1463	0.42	975.48	0.30	17667	0.57
16 Threads - 2 sockets	878.80	0.43	8319	0.64	497.33	0.76	1681	0.42	1009	0.26	22562	0.56

	MPI				OpenMP				UPC			
IS - Size C	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio	Mem traffic (GB)	L3 hit ratio	L3 access $\times 10^6$	L2 hit ratio
1 Threads - 1 socket					32.39	0.03	5600	0.63	44.10	0.07	2629	0.62
2 Threads - 1 socket	47.67	0.13	1846	0.53	32.23	0.06	2817	0.62	45.68	0.08	2463	0.60
2 Threads - 2 sockets	47.52	0.13	1800	0.53	33.84	0.03	5833	0.62	45.75	0.07	2771	0.61
4 Threads - 1 socket	50.99	0.13	2115	0.49	32.10	0.06	2833	0.62	49.57	0.06	3900	0.56
4 Threads - 2 sockets	50.80	0.13	2092	0.50	33.33	0.05	3867	0.62	49.50	0.07	3523	0.57
8 Threads - 1 socket	56.27	0.10	3280	0.46	32.22	0.05	3460	0.62	57.78	0.04	8400	0.47
8 Threads - 2 sockets	55.68	0.13	2423	0.46	33.08	0.05	3911	0.62	56.84	0.05	6889	0.49
16 Threads - 2 sockets	60.77	0.13	2815	0.42	33.24	0.04	4550	0.61	62.93	0.07	5143	0.44

the computational performance of four kernels from the NAS Benchmark, both on a single-node system and on a multi-node supercomputer using three different programming models: UPC, MPI, and OpenMP. On the multi-node supercomputer we observed that UPC is almost always inferior to MPI in terms of performance, although UPC scales well to 1024 cores and 64 nodes, the maximum system size used in this study.

From the measurements performed on the single-node system, we observed that by using more cores the performance and the energy efficiency both increase for the four selected kernels on the chosen hardware platform. The conclusion is not the same on the multi-node computer used in our experiments: the energy efficiency, except in one case, is not increasing with higher numbers of cores and nodes.

We would like to highlight that on the single-node system UPC can compete with MPI and OpenMP in terms of both computational speed and energy efficiency. We obtained this conclusion by analyzing the results produced on the single-node system in order to localize the origin of difference in performance. We found that data locality is the main reason for the difference in performance.

Our conclusions about UPC are compatible with results obtained in previous studies, in particular [8], [9]. We confirm that UPC can compete with both MPI and OpenMP in performance on a single-node.

In addition we provided a thorough analysis of the performance by looking at the L3 cache hit ratio, L2 cache hit ratio, memory traffic and L3 access of MPI, OpenMP and UPC. And we studied the results of UPC, MPI and OpenMP, running the selected kernels from the NAS Benchmarks on 2,4 and 8 cores by using one and two sockets in order to show the interest of the trade-off between energy efficiency and performance.

In future work, we will explore hardware accelerators such as Many Integrated Cores (MIC) and GPUs. These accelerators are well-known for being more energy efficient than CPUs for many applications.

Furthermore, it would be interesting to investigate why UPC, MPI and OpenMP differ in their communication pattern. In order to enhance the energy measurements, we aim for a more fine grained approach of measuring the energy consumption. By studying the energy cost of computation, communication between nodes, and between CPU and memory separately, we can suggest improvements to energy consumption both in the user codes and in the UPC compiler and runtime environment.

Acknowledgments

This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671657, the European Union Seventh Framework Programme (EXCESS project, grant n°611183) and the Research Council of Norway (PREAPP project, grant n°231746/F20).

References

[1] "GASNET Official Webpage," last accessed on 17/12/2015. [Online]. Available: http://gasnet.lbl.gov

[2] Milthorpe, Josh et al., "PGAS-FMM: Implementing a distributed fast multipole method using the X10 programming language," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 3, pp. 712–727, 2014.

[3] Shan, Hongzhang et al., "A Preliminary Evaluation of the Hardware Acceleration of the Cray Gemini Interconnect for PGAS Languages and Comparison with MPI," *SIGMETRICS Perform. Eval. Rev.*

[4] El-Ghazawi, Tarek et al., "UPC Performance and Potential: A NPB Experimental Study," in *Supercomputing, ACM/IEEE 2002 Conference*. IEEE, 2002, pp. 17–17.

[5] ——, "UPC Benchmarking Issues," in *Parallel Processing, 2001. International Conference on*. IEEE, 2001, pp. 365–372.

[6] François, Cantonnet et al., "Performance Monitoring and Evaluation of a UPC Implementation on a NUMA architecture," 2003.

[7] Jose, Jithin et al., "Unifying UPC and MPI runtimes: experience with MVAPICH," in *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*. ACM, 2010, p. 5.

[8] Mallón, Damián A et al., "Performance evaluation of MPI, UPC and OpenMP on multicore architectures," 2009.

[9] Shan, Hongzhang et al., "A programming model performance study using the NAS parallel benchmarks," 2010.

[10] Yelick, Katherine et al., "Productivity and Performance Using Partitioned Global Address Space Languages," 2007.

[11] "NAS Benhcmark Official Webpage," last accessed on 18/12/2015. [Online]. Available: http://www.nas.nasa.gov/publications/npb.html

[12] "NAS Benchmark Implemented in UPC," last accessed on 18/12/2015. [Online]. Available: https://threads.hpcl.gwu.edu/sites/npb-upc

[13] Intel, "Intel PCM Official Webpage," last accessed on 18/12/2015. [Online]. Available: https://software.intel.com/en-us/articles/intel-performance-counter-monitor

[14] Coarfa, Cristian et al., "An evaluation of global address space languages: co-array fortran and unified parallel C," 2005.

[15] Wikipedia, "Wikipedia Definition of PGAS - Last accessed on 18/12/2015," 2015.

[16] Marc Tajchman, CEA, "Programming paradigms using PGAS-based languages. Figure used with the courtesy of Marc Tajchman," 2015. [Online]. Available: http://www-sop.inria.fr/manifestations/cea-edf-inria-2011/slides/tajchman.pdf

[17] "Coarray Fortran Official Webpage," last accessed on 17/12/2015. [Online]. Available: https://gcc.gnu.org/wiki/Coarray

[18] "X10 Official Webpage," last accessed on 17/12/2015. [Online]. Available: http://x10-lang.org

[19] "Cray - Chapel Official Webpage," last accessed on 17/12/2015. [Online]. Available: http://chapel.cray.com

[20] "Global Arrays Official Webpage," last accessed on 17/12/2015. [Online]. Available: http://hpc.pnl.gov/globalarrays/papers/

[21] "OpenSHMEM Official Webpage," last accessed on 17/12/2015. [Online]. Available: http://openshmem.org/site/

[22] Berkeley, "UPC Implementation From Berkeley," last accessed on 18/12/2015. [Online]. Available: http://upc.lbl.gov

[23] Intrepid Technology Inc., "UPC Implementation on GCC," last accessed on 18/12/2015. [Online]. Available: http://www.gccupc.org/

[24] "CLANG UPC," last accessed on 18/12/2015. [Online]. Available: https://clangupc.github.io

[25] Bailey, David H et al., "The NAS Parallel Benchmarks," 1991.

[26] UiO, "Taurus SuperComputer Official Webpage," last accessed on 14/04/2015. [Online]. Available: https://tu-dresden.de

[27] "NAS Benhcmark Official Webpage - Problem Size," last accessed on 18/12/2015. [Online]. Available: www.nas.nasa.gov/publications/npb_problem_sizes.html

[28] Wong, Frederick C et al., "Architectural requirements and scalability of the NAS parallel benchmarks," in *Supercomputing, ACM/IEEE 1999 Conference*. IEEE, 1999, pp. 41–41.

[29] D. e. a. Hackenberg, "Hdeem: High Definition Energy Efficiency Monitoring," in *Energy Efficient Supercomputing Workshop (E2SC), 2014*. IEEE, 2014.

[30] F. e. a. Almeida, "Energy measurement tools for ultrascale computing: A survey," *Supercomputing frontiers and innovations*, vol. 2, no. 2, pp. 64–76, 2015.

[31] M. F. e. a. Dolz, "Ardupower: A low-cost wattmeter to improve energy efficiency of hpc applications," in *Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*. IEEE, 2015, pp. 1–8.

[32] Benedict, Shajulin, "Energy-aware performance analysis methodologies for HPC architectures—An exploratory study," *Journal of Network and Computer Applications*, vol. 35, no. 6, pp. 1709–1719, 2012.

[33] E. e. a. Rotem, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, no. 2, 2012.

[34] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *Proceedings of the 2001 international symposium on Low power electronics and design*. ACM, 2001.

[35] G. Contreras and M. Martonosi, "Power prediction for intel xscale® processors using performance monitoring unit events." IEEE, 2005.

[36] K. e. a. Singh, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, 2009.

[37] B. e. a. Goel, "Portable, scalable, per-core power estimation for intelligent resource management," in *Green Computing Conference, 2010 International*. IEEE, 2010.

[38] D. e. a. Hackenberg, "Power measurement techniques on standard compute nodes: A quantitative comparison," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*. IEEE, 2013.

[39] Top Green 500, "The 500 Green - Energy Efficient High Performance Computing Power Measurement Methodology," last accessed on 23/05/2015.