I. INTRODUCTION

The development of current and next generation processor architectures is currently limited by power and thermal constraints. Performance improvement is not achieved, as in the past, by increasing the processor’s frequency clock but by increasing the number of computing elements within a single processor chip. By sharing hardware resources among computing elements processor manufacturers have been able to demonstrate improved processor utilization and power efficiency, though single-thread performance has reduced. This is especially important for next generation high performance computing (HPC): In order to achieve exaflop performance with a overall power budget of 20-25 MW [1], future generation supercomputers will run billions of threads, with O(10000-100000) threads just within a single compute node. How to assign hardware resources to these threads and how to map them to achieve the desired metric is still an open question.

In the petascale era time-to-completion (performance) was the primary metric of interest, thus all efforts were directed to improved performance, no matter the cost (in terms of power and/or energy) nor the efficiency. In the exascale era, instead, other operational scenarios may require to optimized the execution of an HPC application for a different metric. For example, the user could be interested in running applications in a configuration that provides the lowest energy-to-completion or does not exceed a certain power envelop. If two or more configurations are equivalent (e.g. both running within the target power envelop or providing the same time-to-completion), the user may select the configuration that also optimize a secondary metrics (e.g., the configuration with higher performance between two energy-equivalent configurations). In the exascale era, new metrics, such as energy-to-completion, power efficiency or peak/average power, will be as important, if not more, than the classical performance metrics.

Traditional HPC parallel applications have been implemented using a message passing programming model, e.g., MPI, to exchange data and synchronize processes. With thousands of cores/hardware threads in a single node, it makes sense to explore the use of shared memory programming models to reduce the synchronization and communication overhead, at least for the threads that share a physical memory. On the other hand, the use of multiple programming models within the same application (e.g., MPI across nodes and shared memory within a node) makes modern applications more complex which, in turn, may reduce their performance at scale. In this paper we analyze the performance efficiency of parallel applications that use distributed, shared memory, and distributed/shared memory programming models on modern multi-core/multi-threaded processor architectures. We seek answers to questions such as “Does it pay off to spend effort in using distributed and shared memory programming model in the same applications?”, “What is the configuration that provides the lowest energy-to-completion?”, or “What is the configuration that provides the highest performance while respecting a specific power envelop?”. To answer these questions we consider two very different architectures: the first is a “classical” multi-core x86 architecture where cores are hierarchically distributed in different NUMA domains (AMD Interlagos [2]). The second is a Simultaneous Multi-Threaded (SMT) architecture where hardware threads in the same cores share a considerable amount of hardware resources (IBM POWER7+ [3]–[5]). Both architectures are commonly used in current HPC, DARPA and data center systems and achieve power efficiency by sharing hardware resources among cores/hardware threads. To the contrary of the classical symmetric multi-processing layout (SMP), in which all processing elements were at the same level, in modern architectures cores and hardware threads are clustered into larger computing elements and are hierarchically connected to other cores/hardware threads in the systems. In general, cores/hardware threads in the same cluster share more hardware resources than computing elements in different clusters, which reduces synchro-
I. RELATED WORK

As processor architectures have reached the power wall, many studies have analyzed applications characteristics in different domains. Initiative, such as the Green500 list [6], or benchmark suites, such as SPEC Power [7], demonstrated the high level of interest (and concern) around power and energy efficiency of next generation supercomputers.

Several studies analyzed performance of HPC benchmarks on different architectures. Chan and Yang [8] compare performance of MPI and OpenMP versions of NPB benchmarks (input class A and B) on a AMD Phenom quad-core system. Wu and Taylor [9] characterize the performance of MPI+OpenMP version of BT and SP and compare it with their equivalent MPI and multi-zone versions on BlueGene/P and Cray XT4/5. Chandramowlishwarany, Madduri and Vuduc [10] describe their effort to characterize and tune a fast multipole method (FMM) application on several CMP/CMT systems (Intel Harpertown, AMD Barcelona, and Intel Nehalem). Inoue and Nakatani [11] study the effect of multi-process and multi-thread programming model on CMP/CMT processor for Java benchmarks and PHP applications. The authors conduct their experiments on Sun Niagara T1 (8 cores, 4 way SMT) and on Intel Nehalem (4 cores, 2 way SMT) and conclude that, while core scalability for the two programming models is comparable, SMT scalability is higher for the multi-thread programming model mainly because of the difference in the number of data TLB misses. Interestingly, the authors discover that, unless the memory allocator is aware of the threads’ placement, the SMT scalability of the multi-thread programming model reduces with the number of cores. Stanisic et al. [12] analyze HPC applications running on an ARM system at the base MontBlanc project [13]. These work only characterize the performance implication of using different programming models and/or multi-core architectures. Our study analyzes not only performance but also power consumption and efficiency and energy-to-completion for all applications on both systems. We also use a broader set of applications from different benchmark suites. Finally, we evaluate architectures (IBM POWER7+ and AMD Interlagos) that had not been previously evaluated, at this level of details, in this field.

Other works extract power and energy profiles of applications in HPC or data center environments. PowerPack [14] proposes an out-of-band interface to measure power from various external power meters. The authors use their tool to extract power and energy profiles of NPB benchmarks with various processor frequencies. In this work, we move a step forward and analyze applications with different programming models, applications that use multiple programming models at the same time and the reasons why certain configurations yield higher power or energy efficiency than others. Moreover, we analyze two platforms and the available hardware configurations (e.g., POWER7+ SMT modes). In [15] the authors analyze performance and energy consumption of an irregular NP-complete problem (the Traveling-Salesman Problem) on various platforms and show that applications able to fully use the resources of a many-core achieve higher performance and lower energy consumption than low-power and general-purpose multi-core processors. Irregular applications are fundamentally different from the applications analyzed in this work. First, system running irregular applications are usually oversubscribed (i.e., they run more threads than the number of available cores/hardware threads) whereas system running HPC applications usually run one thread per core/hardware

This paper makes the following contributions:

- An evaluation of performance, power and energy consumption of scientific applications from various benchmark suites on modern multi-core/multi-thread computer architectures.

- Comparison and analysis of distributed, shared and distributed/shared memory implementation of the same algorithms and their impact on performance, power and energy.

- Performance, power and energy comparison of two different computing architecture (AMD Interlagos and IBM POWER7+).

The rest of this paper is organized as follows: Section II reviews related work. Section III describes our platforms and benchmarks. Section IV presents our experimental results for both architectures. Section V concludes this paper.
thread. Second, synchronization is much tighter in HPC applications. As a result, adding additional cores or hardware threads does not always result in higher performance or power efficiency whereas irregular applications are limited by the amount of available memory but can usually use all available cores and hardware threads.

Finally, several work focus on power modeling of multi-core systems [16]–[19] and use the resulting models to characterize the power profile of parallel applications. The work presented in [17], [18], in particular, focus on HPC applications but does not analyze the impact of different and possibly concurrent use of programming models on performance, power and energy consumption, nor analyze the reasons why some configurations provide higher power efficiency than others.

III. EXPERIMENTAL SETUP

This Section briefly describes the processor architectures, the IBM POWER7+ and the AMD Interlagos, and the applications used in this work.

IBM POWER7+ The POWER7+ [3], [4] is an 8-core design with each core (chiplet) having up to 4 SMT threads. Each POWER7+ core is out-of-order, which maximizes instruction-level parallelism (ILP), and features 12 execution units (including 2 fixed point, 2 load/store and 4 floating point units). Each chiplet contains a 32KB 4-way set associative L1 I-cache and a 32KB 8-way set associative L1 D-cache, a private per-core 256KB L2 cache and a portion of the shared 80MB on-chip L3 cache. The L2 is fully inclusive of both the local D/I L1 caches. The POWER7+ “Fluid” L3 cache exploits embedded DRAM technology to maximize area and power efficiency. The L3 cache is, at the same time, private to each core and shared: each core has access to a private, fast portion of the L3 cache but can also access other core L3 sections, if needed. The processor can be configured to run in TurboCore, in which POWER7+ utilizes up to 4 cores per chip with twice the local region cache per core (2x10MB). Computing intensive applications with limited TLP and with high cache locality benefit from this mode as their single thread performance increases. In MaxCore mode, instead, POWER7+ provides up to 8 cores and 32 hardware threads. This mode is intended for highly parallel workload, such as transaction systems. Each core can run in single-thread (ST or SMT1) mode, SMT2 (two threads executing on a core concurrently) or SMT4 (three or four threads executing on a core) mode. For capacity computing (i.e., multiple independent, serial job running in parallel), both SMT2 and SMT4 modes are expected to provide benefits. For capability computing (i.e., parallel applications with high degree of parallelism), SMT4 may not show extra benefits [3].

We performed our experiments on a IBM POWER7+ blade (710 Express) equipped with 8 cores (32 hardware threads) running at 4.2GHz and 16 GB of RAM. In the following experiments when the number of total tasks/threads is 2, 4 the processor is in TurboCore mode and in SMT1 mode; When using 8 tasks/threads, the processor is in MaxCore/SMT1 mode; With 16 tasks/threads, the processor is in MaxCore/SMT2 mode; Finally, with 32 tasks/threads the processor is in MaxCore/SMT4 mode.

AMD Interlagos The second processor architecture tested in this work is an AMD Opteron 6272 (Interlagos) [2]. AMD Interlagos processors are Clustered Multi-Threaded (CMT) chips divided into 2 Multi-Chip Modules (MCMs), and each MCM consists of 4 Clustered Integrated Cores, a shared 8MB L3 cache, and a memory controller. A Clustered Integrated Core (CIC) couples 2 “conventional” x86 out-of-order Integer Cores together; the two cores share fetch and decode units, the 64KB L1I and the 2MB L2 cache (the 64KB L1D is private to a single Integer Core), and the FP unit. A CIC is midway between a dual-core and a 2-way Symmetric Multi-Threaded (SMT) design. HPC applications heavily use the floating point unit, hence, from an HPC point of view, a Clustered Integrated Core is more similar to a 2-way SMT design.

The AMD Interlagos system consists of two AMD Opteron 6272 sockets, for a total or 32 cores, and is equipped with 64 GB of RAM divided in four NUMA domains. We allocate MPI tasks to maximize performance, hence for n=2 we schedule one task per socket; for n=4 we allocate one task per MCM: for n=8, 16 one task per CIC (one or two tasks per MCM, respectively); finally, for n=32 all cores are used.

Applications For both systems we compiled the applications with GNU GCC (gcc 4.4.7 for the POWER7 system and gcc 4.7.2 for the Interlagos system) for 64-bit architectures with all available optimization (-03, -fopenmp where applicable, loop-unrolling, etc.) and linked to OpenMPI library (OpenMPI 1.6.5 for the POWER7 system and OpenMPI 1.6.3 for the Interlagos system). NPB applications [20] are taken from the NPB 3.3.1 and NPB 3.3-MZ benchmark suites and use the class C input set, while the Sequoia [21] (AMG and SPHOT) use their respective reference input sets. We explicitly deactivate the unused cores/hardware threads through the /sys/devices/system/cpu/cpuX/online interface to reduce the idle power and minimize hardware resource contention. Finally, we statically bind each MPI tasks or OpenMP threads to a particular hardware threads.

Power Instrumentation Power consumption for both systems is measured using a WattsupPro [22] external power meter that provides a power sample every second. Although this resolution is not particularly high, the applications examined in this work are fairly regular and their power consumption is stable. Moreover, we use large data sets for all applications and the running time is generally above 5 minutes (except for IS, which only runs for 45 seconds), thus we are able to collect a sufficient number of power samples for our statistics.

The AMD Interlagos processor also features a processor power sensor with a 1ms resolution [2]. The power sensor is similar to the one provided by Intel SandyBridge processors [23] but does not require superuser privilege to access power information, as it is, instead, required to access Intel MSR. The AMD Interlagos processor sensor provides a higher accuracy than the WattsUP external power meter but does not provide information about the memory and the uncore components of the system.

In order to provide a apple-to-apple comparison, we use Wattsup external power meters for both systems (which also include memory power). However, we validated the power samples obtained from the Wattups power meters against the power readings obtained from the AMD processor power sensor. Our experiments show that the values obtained for the total energy consumption computed with the WattsUP external power meter and the AMD Interlagos processor power sensor are comparable and within 5% of error. Since in this paper we mainly show power and energy ratios and dynamic power and energy consumption, we believe that this error is tolerable.
IV. EXPERIMENTAL RESULTS

In this section we analyze performance, active power, energy and power efficiency of parallel applications on both IBM POWER7+ and AMD Interlagos using distributed (MPI), shared memory (OpenMP) and distributed/shared memory (MPI/OpenMP) programming models.

The experiments reported in this section are the average of 10 (no outliers was removed). Speedup is computed as the ratio between the sequential and the parallel execution time. Power is measured with external WattsUP power meters. In this work we focus on active power and energy consumption, measured as the difference between the application’s power consumption and the system idle power or static power (measure with the external power meter when the system is idle) at every sample. The static power is constant and does not depends on the particular activity performed by an application, hence the total power consumption will be the sum of active and static power consumptions. An analysis of the static power and the techniques used to reduce it is out of the scope of this work, but extensive documentation is available in the literature [1].

In the following experiments we report the average active power for each applications. Active energy is computed integrating the active power curve over time with classical trapezoidal methods. Since the applications show a stable power consumption, the standard deviation from the average results (performance, power and energy) is contained.

Analysis of MPI Applications: In our first experiment we analyze performance power and energy consumption of pure MPI applications, historically used in distributed systems.

Figures 1 and 2 show our results for the MPI version of NPB applications when varying the number of MPI tasks from 2 to 32. In this experiment, and in the rest of the paper, we do not show results for BT and SP with 2, 8 and 32 concurrent threads, as these applications require a perfect squared number of MPI processes (4 or 16) to run correctly.

MPI applications (Figure 1a) generally scale well up to 8 cores on the IBM POWER7 system (SMT1 mode) with EP showing linear scalability and CG and LU showing good scalability (above 4x). Using the secondary (SMT2) and tertiary (SMT4) hardware threads does not usually provide extra performance (except for EP and marginally for BT, IS and LU in SMT2 mode); in some case (CG and MG) the hardware resource contention of the STM modes reduces performance. To verify that hardware resource contention is the cause of the limited performance, we studied the efficiency of NPB applications in terms of operations/second. Our goal is to quantify the single thread performance degradation caused by sharing resources with additional cores/hardware threads and to understand if the extra thread-level parallelism (TLP) is enough to provide overall performance improvement. More in details, Figure 1d shows that the number of operations/second/thread generally decreases when using additional MPI processes with respect to single thread execution. This is partially due to the overhead of communication in which a parallel application incurs but it is also caused by hardware resource contention. As we can see from the graph, there is a significant performance drop beyond 8 parallel MPI processes, when the secondary and tertiary hardware threads are used. If the extra TLP makes it up for the loss in terms of operations/second/thread, the
application will still experience a net gain. If the single-thread performance degradation is higher than the TLP, increasing the number of parallel threads will not increase performance and may eventually lead to performance degradation. Figure 1e shows that the aggregated number of operations/second always increases up to 8 cores, which means that the applications benefit from the higher level of parallelism even if the per-thread performance decrease, as also shown in Figure 1a. When the secondary and tertiary hardware threads are activated (more than 8 tasks), applications’ speedups decrease considerably and may result in performance degradations. A STM2 and SMT4 modes, the amount of hardware resources shared among the threads running in a core increases. Hardware threads in the same core not only share the L3 cache and the memory bandwidths, but also share the core functional units, the fetch bandwidth, the L1, L2 and TLB caches. In some cases (FT and MG) per-thread performance reduces by more than 1/2 when moving from STM1 to SMT2 and from STM2 to SMT4. This is a hint that the extra TLP (double the number of computing elements) might not be sufficient to provide overall performance improvements. FT, MG and LU are the most computing intensive applications, as the high number of operations per second per thread in Figure 1d shows, thus it is normal to expect a higher impact when turning on the secondary and tertiary hardware threads. The configurations that achieve the highest performance are usually the ones with 8 (CG, FT, MG) or 16 (JS, LU) MPI tasks, excepts EP, which achieves the highest performance with 32 MPI tasks. There is, thus, a “sweet spot” in which applications should run to achieve maximum performance (time-to-completion).

In the exascale era, however, performance will not be the only important metric; other metrics, such as the energy required to complete an application (energy-to-completion), the power efficiency or the peak power consumption, will be as important as the classical “time-to-completion” metric. Figure 1b shows that using additional hardware threads increases the system average active power consumption. If combined with performance improvements, this increase in power may still provide energy savings (Figure 1c), which may lead to lower total energy to run an application (which directly translates in saving money for the electricity bill). However, if the increased power consumption is not coupled with a level of performance (execution time) that makes it up for the extra active power, the application will consume more energy than an configuration with a lower number of parallel tasks. In the worst-case scenario (high power consumption and performance degradation), the extra energy spent to run an application with higher number of parallel threads may be completely wasted, as the application will take longer to complete and consume more power. Figure 1b shows that adding additional MPI tasks often increases the applications’ active power consumption. However, the active power increases are generally sub-linear with the increase number of parallel tasks. When using secondary and tertiary hardware threads, the active power increase is minimal (which is one of the main reasons why SMT processors are designed). In some cases (FT and MG with 32 MPI tasks), performance degradation is such that processor’s activity globally reduces, hence the active power consumption becomes lower than configurations with fewer parallel tasks. In a scenario in which an HPC system runs in a power-constrained environment, the configurations that
Figure 2b shows that average active power consumption increases with the core activity (aggregated operation/second) up to 16 MPI tasks. When sharing the FPU, the number of stalled cycles increases, thus core activity decreases. The result is that the active power consumption with 32 MPI tasks is lower than in configurations with 16 tasks. For this architecture, however, the increased power consumption is coupled with higher performance, which results in lower energy-to-completion when increasing the number of MPI tasks. Figure 2c shows that the configurations with lower
These experiments show that, as long as the additional TLP makes it up for the per-thread performance degradation, the total number of operations per second increases and the applications are able to use, to some extent, the extra processing elements. If the total number of operations per second decreases, the performance degradation experienced by each application’s task may overshadow the benefit of extra computing power. This does not necessarily mean that the application will not scale but the speedup provided by the extra cores/hardware threads might be limited. In these scenarios, adding extra processing elements may not be worth, especially if other metrics, such as peak power or energy-to-completion, are considered. In these situations using extra cores/hardware threads to perform auxiliary operations (such as memory prefetching or speculation) may provide higher performance [24], [25]. For example, secondary and tertiary hardware threads on POWER7+ or secondary could be use to run data analytics applications next to the scientific simulation that produces the data, thereby avoiding the need to move data from the computing node memory to storage and then back to the computing node memory when running the data analytics.

Comparing the two architectures, we notice that the performance degradation suffered when activating SMT2 and SMT4 modes in POWER7+ is much larger than the multi-core mode in the AMD Interlagos: despite the lower clock frequency, the Interlagos system provides almost double the performance (in terms of total number of operations per second) when using all 32 computing elements. Based on the active energy, the AMD Interlagos system is also more “energy efficient” that the POWER7+ system: using additional MPI tasks generally reduces the energy-to-completion of HPC applications, whereas for POWER7+ there is a “sweet spot configuration” that uses less than the total number of hardware threads. 

Analysis of OpenMP Applications: Within a single node computing elements can directly access to the same physical memory, thus it becomes appealing to use shared memory programming models, such as OpenMP, to reduce synchronization overhead. In this section we analyze the effects of using a shared memory programming model, OpenMP in our case, on the two platforms. Figures 3a and 4a show the speedup of OpenMP NPB applications with respect to serial version for IBM POWER7+ and AMD Interlagos, respectively: Compared to their equivalent MPI implementations, the OpenMP versions of NPB applications generally show higher speedups. With MPI, in fact, processes communicate through message passing, which involves the use of the network stack and, in Linux-based systems, system call context switches. Communications through shared memory, on the other hand, is much faster because memory locations can be directly accessed (though synchronization mechanisms, such as locks, may increase the access latency).

Figure 3a also shows that the use of the secondary and tertiary hardware threads in POWER7 does not generally introduce performance degradation (except for MG and SP), though it does not provide performance improvement ei-
For the AMD Interlagos system, using OpenMP (Figure 4a) generally provides higher speedups than using MPI (Figure 2a). Applications such as EP show a significant performance improvement when using 32 OpenMP threads, while SP is still limited by the floating point unit and L3 contention. Figure 4d reports the number of operations per second per thread for this system: As for the MPI implementation, the graphs show that there is considerable single-thread performance degradation when using the second core in a CIC (32 threads). However, the higher level of TLP makes it up for the lower single-thread performance, thus the aggregated number of operations/second increases up to 8 cores and then flattens beyond that (except for MG and SP, for which we observe performance degradation) even if single thread performance reduces when sharing hardware resources (Figure 3d). The active average power consumption increases when using additional OpenMP threads but, again, less than linearly with the number of OpenMP threads, especially when using the secondary and tertiary hardware threads. The resulting power efficiency show a constant trend (again, MG and SP are exceptions) thus the energy-to-completion does not generally vary when increasing then number of OpenMP threads. For CG, BT, EP, FT, IS, and LU there are several energy-equivalent configurations, which also correspond to performance-equivalent configurations. For MG and SP the configuration with the lowest energy-to-completion is the one with the lowest number of concurrent OpenMP threads. On the other hand, for EP and LU the configuration with the lowest energy-to-completion is the one with 32 OpenMP threads.

Even for the AMD Interlagos system, using OpenMP (Figure 4a) generally provides higher speedups than using MPI (Figure 2a). Applications such as EP show a significant performance improvement when using 32 OpenMP threads, while SP is still limited by the floating point unit and L3 contention. Figure 4d reports the number of operations per second per thread for this system: As for the MPI implementation, the graphs show that there is considerable single-thread performance degradation when using the second core in a CIC (32 threads). However, the higher level of TLP makes it up for the lower single-thread performance, thus the aggregated number of operations/second increases for all applications, except SP, up to 32 OpenMP threads. Interestingly, the power efficiency (Figure 4f) increases when using all 32 cores (except for SP which shows power efficiency degradation beyond 4 OpenMP threads). In some cases, (LU, BT, and FT) the increase is considerable. These results show that the configurations with the lowest energy-to-completion are generally the ones that use 32 OpenMP threads, except for SP for which using 32 threads provides an energy-to-completion value that is 3 times the energy-to-completion with 4 cores and lower performance.

Analysis of MPI+OpenMP Applications: As we showed in the previous Section, parallelizing application through shared memory programming models generally yields better results than message passing. However, supercomputers consist of thousands of nodes that do not share a common memory, thus, a form of message passing (although not necessarily MPI) is still required for communication that involves processes on different nodes. A possible solution consists of using message passing across nodes and shared memory within a node. This approach would force programmers to parallelize their applications using (at least) two different programming models, however, the gains seem worth it. Alternatively, distributed shared memory, such that of Tang et al. [26], or Partitioned Global Address Space [27] approaches provide the
programmer a global address space across the whole cluster.

In order to characterize the performance implications of running applications parallelized with different programming models at the same time, we examine the performance of applications from the Sequoia benchmark suite [21] and from the multi-zone NAS Parallel Benchmark suite that are parallelized with MPI and OpenMP. A “natural” operational scenario consists of running one MPI task per compute node and several OpenMP threads inside each node, one for each processing element (core or hardware thread). In this scenario, communication between different compute nodes is performed through MPI primitives while the OpenMP threads inside a node use shared memory to exchange data. However, because of the way hardware resources are clustered together, this scenario might not necessarily be the optimal one.

Figure 5a and 6a show the scalability of NPB-MZ applications, AMG2006 and SPHOT\(^1\) from the LLNL Sequoia benchmark suite when varying the number of MPI tasks \(N\) and the number of OpenMP threads per MPI task \(M\), where \(N \times M = 32\) (i.e., the maximum number of cores/hardware threads available in our systems). As we can see from the graphs in Figure 5a and 6a, although the applications use always the same number of cores (32), performance varies, in some case (such as BTMZ and SPMZ) quite considerably.

Note that Lum-MZ does not run with 32 MPI tasks or with 32 OpenMP threads in class C. The graphs show that all applications provide higher performance when using both MPI and OpenMP within a single node, i.e., communication through shared memory helps but, in some cases, it may create higher resource contention. For the IBM POWER7+ system (Figure 5a), the highest performance is achieved when using either 16 hardware threads (BTMZ, SPMZ) or the highest number of OpenMP threads per MPI tasks (2x16 for LUMZ and 1x32 for AMG2006). In order to understand the reasons why there is such performance difference among the tested configurations, we analyzed other performance metrics extracted from processor performance counters, such as TLB misses, the last-level cache misses, stalled cycles, etc. Figure 7a reports that the number of data TLB (DTLB) misses is inversely correlated with the speedup, i.e., the higher the number of TLB misses, the lower the performance. Although TLB entries are shared only among hardware threads in the same core, modifying the application’s memory mapping (e.g., dynamic memory) may invalidate the virtual-to-physical mappings stored in the core’s TLB and force the OS to invalidate all the TLB entries related to the same address space in all the other cores (TLB shootdown). Our experiments show that using only OpenMP threads may generate a high inter-core traffic, which might limit performance. For IBM POWER7+, using fewer than 4 OpenMP threads per MPI task and mapping all threads belonging to one MPI task to the same core minimizes the effect of TLB misses. On the other hand, using 2 or more MPI tasks on the same core implies that each MPI process will use half the available TLB entries to map its address space, which may increase TLB misses. The optimal case is thus 8/16 MPI tasks (1/2 per core), each with 4/2 OpenMP threads. In this scenario threads inside the same core still benefit from sharing TLB entries while the number of TLB shootdowns is limited, as threads running on different cores do not share the same address space. Average active power also varies among the different configuration and usually follows performance (i.e.,

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LLC misses</th>
<th>DTTLB misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTMZ</td>
<td>1.5e+08</td>
<td>5e+09</td>
</tr>
<tr>
<td>LUMZ</td>
<td>2.5e+08</td>
<td>1e+10</td>
</tr>
<tr>
<td>AMG2006</td>
<td>1e+08</td>
<td>2e+10</td>
</tr>
</tbody>
</table>

\(^1\)We will add the experiments for SHOT on POWER7+ in the final version of the paper.

For the AMD Interlagos system, using a small number of OpenMP threads per MPI process generally provides higher performance (all applications except LUMZ). The reason is that this system consists of four NUMA domains and modifying memory locations shared across NUMA domains may increases the number of messages exchanged on the HyperTransport bus to guarantee memory coherence in the L3 cache. Using more than 4 MPI tasks and carefully mapping them to different NUMA domains, instead, eliminates the need of maintaining coherency across the NUMA domains. We also observe that the number of last-level cache (LLC) misses varies from one configuration to another and that there is a correlation between the number of LLC misses and the performance obtained in a specific configuration (Figure 7b). This is especially true in configurations with more than 1 MPI per MCM. The average active power consumption (Figure 6b) is generally stable across the different configurations, with the exception of AMG2006 and SPHOT, which show a lower active power with 1 MPI tasks and 32 OpenMP threads. This means that the energy-to-completion is mainly a function of performance, hence the configurations with the lowest energy-to-completion values are the ones with 1 or 2 OpenMP thread/MPI task.

V. Conclusions

In this paper we have analyzed performance, power and energy consumption of scientific HPC applications pro-
programmed with MPI, OpenMP or MPI+OpenMP on two different architectures, IBM POWER7+ and AMD Interlagos. While both architectures share hardware resources among cores/hardware threads, their approaches are radically different. IBM POWER7+ uses symmetric multi-threading technology to reduce the memory latency while the AMD Interlagos is a multi-core system where pairs of cores floating point unit and the L2 cache.

Although our results confirm using shared memory to synchronize threads generally provides higher performance, we found out that using only shared memory within a single node does not provide the highest performance and power efficiency. Instead, a combination of MPI tasks and OpenMP threads is required to achieve high performance and efficiently on both architectures. From a power/energy point of view, our experiments show that the AMD Interlagos system generally provides the lowest energy-to-completion when using all computing elements while for POWER7+ there is a “sweet spot” which consist of less than the total number of computing elements. Moreover, we showed that the “best configuration” strongly depends on the target metric(s) and the underlying architecture. We believe that our experience can help understanding current and future scientific applications and can guide the development of adaptive software algorithms that can maximize different metrics.

ACKNOWLEDGMENTS

This work was supported by the DOE Office of Science, Advanced Scientific Computing Research, under award number 62855 “Beyond the Standard Model – Towards an Integrated Modeling Methodology for Performance and Power”; Program Manager Karen Pao.

REFERENCES


