Characterizing the Impact of Using Spare-Cores on Application Performance

José Carlos Sancho[1], Darren J. Kerbyson[2], and Michael Lang[3]

[1] Barcelona Supercomputing Center, Barcelona 08034, Spain
[2] Pacific Northwest National Laboratory, Richland, WA 99352, USA
[3] Los Alamos National Laboratory, Los Alamos, NM 87545, USA
jsancho@bsc.es, Darren.Kerbyson@pnl.gov, mlang@lanl.gov

Abstract. Increased parallelism on a single processor is driving improvements in peak-performance at both the node and system levels. However achievable performance, in particular from production scientific applications, is not always directly proportional to the core count. Performance is often limited by constraints in the memory hierarchy and also by a node inter-connectivity. Even on state-of-the-art processors, containing between four and eight cores, many applications cannot take full advantage of the compute-performance of all cores. This trend is expected to increase on future processors as the core count per processor increases. In this work we characterize the use of spare-cores, cores that do not provide any improvements in application performance, on current multi-core processors. By using a pulse-width modulation method, we examine the possible performance profile of using a spare-core and quantify under what situations its use will not impact application performance. We show that, for current AMD and Intel multi-core processors, spare-cores can be used for substantial computational tasks but can impact application performance when using shared caches or when significantly accessing main memory.

1 Introduction

Increased silicon integration is the driving force behind the rapid increase in the number of cores available on a single processor. State-of-the-art mainstream processors from AMD, IBM and Intel already boast between 8 and 12 cores in a single processor, and many more cores are foreseen in the future such as the Intel 48-cores processor [1]. However the achievable performance, especially when considering scientific applications, is not always directly proportional to the core count. Many applications can only take advantage of a sub-set of the available cores before the achieved performance peaks [2]. Thus resulting in two classes of cores – those dedicated for application use, and those deemed as *spare-cores*.

A significant factor for many applications is the performance of the memory sub-system and in particular the available main-memory bandwidth. This is currently determined by the available pins dedicated to I/O in integrated circuits (IC). This fully depends on the IC manufacturing process. In today's IC

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part I, LNCS 6271, pp. 74–85, 2010.

processes such as Very-Large Scale Integration (VLSI) and some Ultra-Large Scale Integrations (ULSI) like the System-on-a-chip (SoC), the number of pins is proportional to the IC perimeter. Unfortunately the perimeter is not increasing significantly due to cost constraints. On the other hand, there are some ULSI proposals such as the utilization of Through Silicon Vias (TSV) [3] that promises to substantially increase the available bandwidth between processor and memory. This is achieved by stacking ICs on top of each other, but is still in experimental stages. In the short-term we expect that there will be an increase in the available spare-cores on a processor given the rise in the overall core-count.

There have been many proposals for using spare-cores in support activities [4], [5], [6], [7], [8], and [9], including to increase reliability or to monitor other core's activities, but little has been done to characterize what spare-cores can actually do from a performance standpoint. Even though spare-cores may be available, they share common resources available to all cores within a processor typically including: a level of shared cache, shared memory controllers (providing access to main memory), and shared network-interface-controllers. In this work we characterize the performance profile of using a spare-core in terms of its impact on application performance.

A Pulse-Width-Modulation (PWM) approach is used to characterize the impact of using the spare core. Two phases, an active and an inactive, exist within a single cycle which is continuously repeated during application execution. A separate micro-benchmark is used for each of: compute, local-cache access, shared-cache access, main-memory access, and inter-node network access; that incorporates the PWM approach. Though other approaches can be used PWM corresponds to a typical use of a spare-core running a process which exhibits phases such as a compute-phase, followed by a data-flush phase (to main memory), and a data-storage phase (to remote disk) for instance.

This work characterizes the impact of using spare-cores on two state-of-the-art processing nodes: a four-processor six-core node from AMD (Istanbul), and a four-processor eight-core node from Intel (Nehalem). Four applications, which are either compute-bound or memory-bound, are used in this analysis. The impact on performance of using a spare-core, as we will see, is application dependent – for instance memory intensive applications are significantly slowed by spare-cores using main memory. The contributions of this work are threefold, first we present a methodology that uses PWM to represent the activities of spare-cores; second we characterize what operations spare cores can actually perform without significantly impacting application performance, and finally we investigate and quantify the approach on current state-of-the-art processing nodes.

The rest of this paper is organized as follows. Section 2 describes our PWM approach as well as detailing the test-bed nodes used. Section 3 details the applications used and their achievable performance on the test-beds. Section 4 characterizes the impact on application performance of using spare-cores and discuss under what situations their use may be appropriate. Related work on analyzing the performance impact of using spare-cores is summarized in Section 5. Conclusions from this work are given in Section 6.

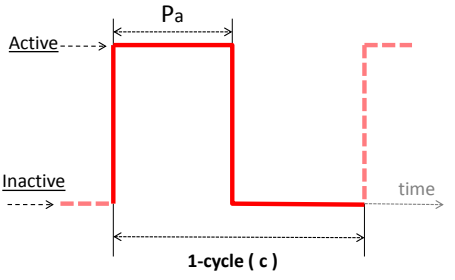

Fig. 1. Pulse-Width-Modulation approach

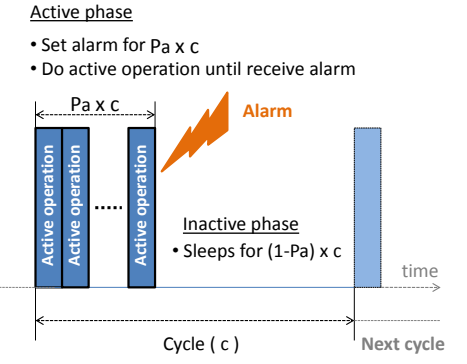

Fig. 2. Implementation of the Pulse-Width-Modulation for using spare-cores

2 Performance Characterization Approach

2.1 Pulse-Width-Modulation Method

The method to characterize the use of spare-cores is based on Pulse-Width-Modulation (PWM). In this commonly used waveform there are two phases: an active and an inactive phase, which are continuously repeated. The width of the active phase can be varied to be between 0% and 100% of the cycle as shown in Figure 1. Operations are performed on the spare-cores in the active phase of the PWM approach while concurrently the application is run on the other cores. The results of the concurrent runs are then compared to the peak performance achievable for the application in isolation to characterize the performance impact of using the spare cores under varied loads.

The implementation of PWM for use on the spare-cores is depicted in Figure 2. The waveform of the PWM is implemented using a software interrupt. At the start of a cycle an interrupt is scheduled at time $P_a \times c$ in the future where c is the width of a cycle (in seconds) and P_a is the proportion of a cycle, between 0 and 1, that the operation is active, therefore the interrupt determines the end of the active portion of the cycle. Cycle widths of 1, 5, and 10 seconds have been considered in this paper. During the active interval, a particular operation is performed multiple times until the software interrupt occurs. The software interrupt is implemented using a chain of software interrupts in order to deliver a signal in both second and microsecond granularities. The Linux functions *alarm* (seconds) and *ualarm* (microseconds) are used for this purpose. These functions send the software interrupt *SIGALRM* to the current process. The signal handler temporarily stops the current PWM operation— puts this PWM process to sleep for a time, $(1-P_a)\times c$, until the next cycle. Note that during the inactive interval the PWM scheme will allow the Operating-System to use these cores as needed for their own activities without interrupting the cores running the application.

In this work we use five operations that are independently active during each PWM cycle on the spare-core. The details of each operation are as follows:

- **Compute:** The active operation consists using the functional units of the spare core and registers without use of any shared resources.
- **Private-cache access:** The active operation walks a data buffer of size equal to that of the L2-cache of the processor-core.
- **Shared-cache access:** Similar to the private-cache access, the active operation walks a data buffer of size which is now equal to that of the shared L3-cache of the processor.
- **Main memory access:** Similar to the cache access, the active operation walks a large data buffer, in this work was set to be 100MB, which when used requires access to main memory.
- **Inter-node network:** The active operation consists of a the repeated communication of a data buffer between spare-cores located on different nodes. In this work was set to be 1MB.

Each is considered independently to represent the typical expected use of spare-cores. The operations performed will be in phases, such as processing in a compute-phase, repeatedly using private buffers which may be cache resident, transferring private buffers to main memory, and also storing information on remote disks accessible through the inter-node network.

In addition, when using more than one spare-core in the testbeds, the operations from all of these spare-cores are synchronized together at the beginning of each cycle in the PWM scheme. This synchronization is performed using the *MPI_Barrier* primitive. This synchronization is required in order to minimize the impact of the PWM processes which could act like *Operating System noise* [8] and interfere with synchronization phases of the application if they were allowed to be run as asynchronous independent processes.

2.2 Test-Bed Systems

The PWM characterization approach is applied to two current state-of-the-art multi-core nodes from AMD and Intel in this work. The first contains four 6-core AMD Istanbul processors and the second contains four Intel Nehalem processors.

As listed in Table 1, each Istanbul processor, executing at 2.6GHz, has a single memory controller with two DDR2-800MHz memory channels, and a L2 cache and L3 cache size of 512KB and 6MB, respectively. Each Nehalem processor, executing at 2.4GHz, also has a single memory controller but with four DDR3-1333MHz memory channels, and has a L2 cache and L3 cache size of 256KB and 24MB, respectively.

Table 1. Current state-of-the-art multicore processors

Vendor	Name	Clockspeed	Gflops	Cores	L2cache	L3cache	Memory	Channels
AMD	Istanbul	2.6GHz	62.4	6	512KB	6MB	DDR2-800	2
Intel	Nehalem	2.4GHz	76.8	8	256KB	24MB	DDR3-1333	4

3 Application Performance

Four production applications from the Department of Energy are used in this work:

- **SAGE.** An adaptive mesh refinement (AMR) hydrodynamics code used for the simulation of shock-waves which was developed jointly by Los Alamos National Laboratory and SAIC [10].
- **PARTISN.** A deterministic radiation transport code from Los Alamos National Laboratory that solves the Boltzmann equation using the discrete ordinates method, on structured meshes.
- **XNOBEL.** An adaptive mesh refinement (AMR) hydrodynamics code with high explosive calculations for the the simulation of shock-waves which was developed by Los Alamos National Laboratory.
- **SWEEP3D.** A code-kernel from Los Alamos National Laboratory that implements deterministic radiation transport. The computation is in the form of wave-fronts that originate at the corners of a 3-D physical space [11].

The input decks for the applications are summarized in Table 2. For each application a fixed problem size was used. The problem was subdivided across the cores used in each processor, corresponding to a strong scaling mode of operation. The measured performance on both the AMD Istanbul and Intel Nehalem nodes for SAGE, PARTISN, XNOBEL, and SWEEP3D are shown in Figures 3, 4, 5, and 6 respectively. For each experiment the number of cores per processor was varied between 1 and the maximum available on each processor, the performance reported in terms of the processing rate. The processing rate was aggregated over all cores used and reported as cells processed per second. It can be seen that not all of the cores are used productively for all of the applications. Contention for shared resources, such as access to main memory, result in less work being accomplished as additional cores are used. This is especially true for memory intensive codes including SAGE and XNOBEL.

In the case of SAGE, no additional performance arises when using more than 3 cores per Istanbul processor, or when using more than 6 cores per Nehalem processor. The performance of XNOBEL peaks at 4 cores per Istanbul processor and 2 cores per Nehalem processor. In contrast for SWEEP3D, which is compute-bound, we see a very different performance profile. Using more cores per processor results in additional performance, excluding some variability on the Nehalem. SWEEP3D is much less reliant on main memory due to its working-set

Table 2. Application's input decks and processing characteristic

Name	Input deck	Problem size	Processing characteristic
SAGE	*timing_h*	$560,000$	Memory
PARTISN	*Rep1*	$400 \times 24 \times 24$	Memory
XNOBEL	*sc301*	$575,000$	Memory
SWEEP3D	*Pencil*	$40 \times 20 \times 400$	Compute

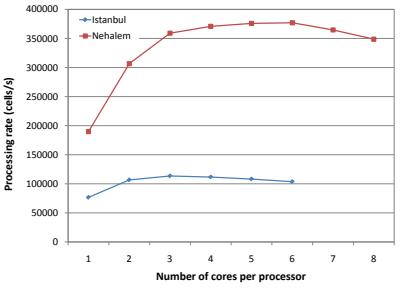

Fig. 3. SAGE performance scaling

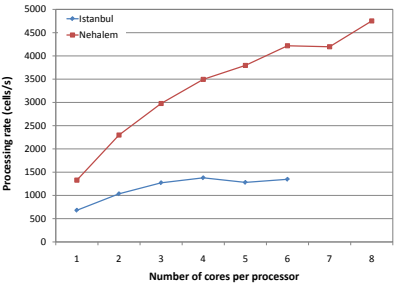

Fig. 4. PARTISN performance scaling

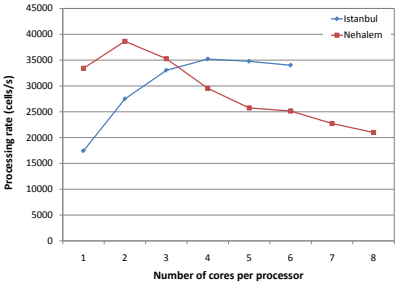

Fig. 5. XNOBEL performance scaling

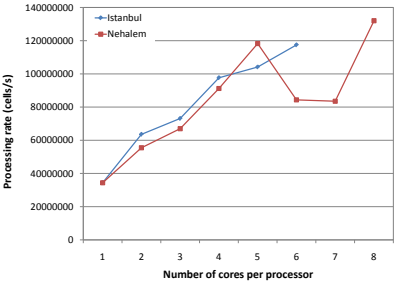

Fig. 6. SWEEP3D performance scaling

fitting mostly into the shared L3 cache. The performance of PARTISN is more complicated in that the result is different across the two nodes. On Istanbul the performance peaks at 4 cores in contrast to Nehalem in which using additional cores always adds performance.

The number of cores that achieves peak application performance is summarized in Table 3 for both nodes under test. Unlike the other applications listed in the table SWEEP3D's performance increased with each additional core used even when all cores in the processor were used. A further interesting case is PARTISN on the Nehalem which also can take advantage of all 8 cores. This is due to the problem size shrinking, through the use of strong scaling, so that less contention was induced on the memory subsystem. Note also that the number of cores that give best performance is application and system dependent.

We will use the peak application performance to compare with the performance achieved when using the PWM on the spare-cores. On both test-beds four spare-cores, one on each processor, are used to run PWM. The rest of the cores are dedicated to run the application or not used at all. Applications only use the core counts that achieve peak performance as listed in Table 3 at except of the applications that can use the full node such as SWEEP3D. For these applications one core is taken from the application to be used by PWM, but the comparisons are made to the application running on the full node fairly

Table 3. Peak application performance (cells/s) and number of cores per processor that achieve that performance on each test-bed

Test-bed	SAGE (cores)	PARTISN (cores)	XNOBEL (cores)	SWEEP3D (cores)
Istanbul	113K (3)	1378 (4)	35K (4)	117M (6)
Nehalem	377K (6)	4752 (8)	38K (2)	132M (8)

representing the full performance cost of using the spare core. In addition, results showed in the next section corresponds to using 1-second cycles for the PWM.

4 Results

The results of applying the PWM approach on a single spare core per processor of the Istanbul node are shown in Figures 7-10 for SAGE, PARTISN, XNOBEL, and SWEEP3D respectively. We show the performance degradation, that is the performance lost when using the spare-core in comparison to the best performance observed (as presented in Table 3 when using the spare-core for each of a compute, private-cache, shared-cache, and main memory operation).

The general observation across all applications is that the degradation of using the spare-core for main memory is very similar to that of using shared-cache, which is much greater than when only using private L2-cache or just performing a computation operation. In addition the impact increases in proportion the active phase of the PWM cycle.

Differences in the performance degradation can also been seen across applications. For the memory-bound code of SAGE, PARTISN and XNOBEL, the use of the spare-core for compute is negligible no matter the duration of the active phase. Since these applications cannot take advantage of all cores, as shown in Section 3, we see that spare-cores can be used for compute tasks without impacting performance. In contrast, Sweep3D is compute-bound, and reducing the number of application cores to create a spare-core significantly reduces application performance. It can be seen that the minimum performance impact is 25% in this case.

When the spare-core uses the private L2-cache we see that the performance degradation on all applications increases with the PWM active phase. This effect can be attributed to the extra traffic generated by the cache coherence protocol. The performance impact is application dependent ranging from 0.3% up to 5.6% for SAGE, and from 2.4% up to 10.6% for PARTISN for instance.

A similar performance impact can be seen when the spare-core uses either the shared L3-cache or uses main memory. Significantly higher performance degradations occur especially as the PWM active phase increases. When the active phase is 100%, the performance degradation reaches 33%, 43%, 30% and 39% for SAGE, PARTISN, XNOBEL and SWEEP3D respectively. This is caused not only by the extra coherence traffic generated, but also by the sharing of the L3 cache bandwidth between the application and the spare-core.

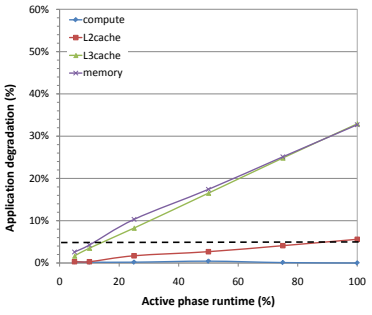

Fig. 7. SAGE degradation on Istanbul

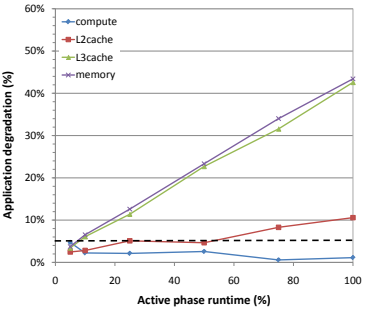

Fig. 8. PARTISN degradation on Istanbul

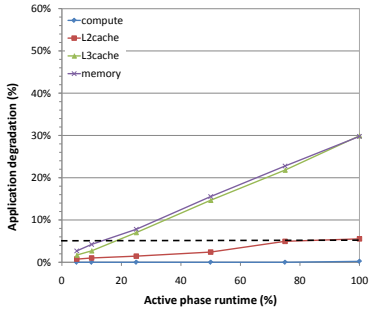

Fig. 9. XNOBEL degradation on Istanbul

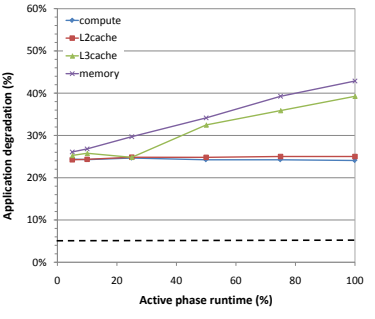

Fig. 10. SWEEP3D degradation on Istanbul

The results of applying the PWM approach on a single spare core per processor of the Nehalem node are shown in Figures 11-14 for SAGE, PARTISN, XNOBEL, and SWEEP3D respectively. The general observations seen on the Istanbul node also apply to the Nehalem node. Though it is interesting to note that PARTISN is compute-bound on the Nehalem node and that using a core as a spare core significantly impacts the application performance in all cases in a similar way to SWEEP3D.

An interesting question to ask is: How much compute, or private-cache, or shared-cache or main memory resources can the spare-core use without impacting application performance? We answer this by considering an *allowable* application performance degradation of 5%. This can be seen for the PWM approach in Figures 7-14 when each of the performance curves reach a 5% performance degradation - as indicated by the horizontal dotted lines. A summary of the PWM active phase durations is listed in the table 4 for all cases - operation type, application, and test node.

For the applications that can take advantage of all cores in a processor the answer to this question is simple - using a core as a spare-core significantly degrades performance in all cases. This can be seen by the zeros in Table 4 for

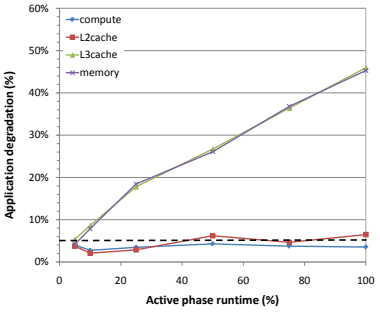

Fig. 11. SAGE degradation on Nehalem

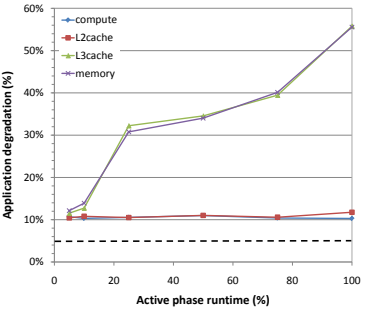

Fig. 12. PARTISN degradation on Nehalem

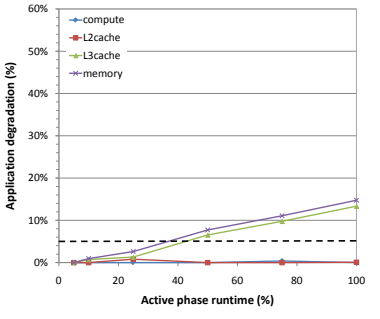

Fig. 13. XNOBEL degradation on Nehalem

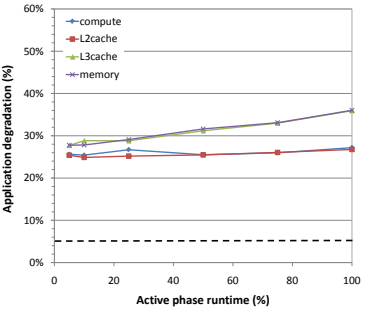

Fig. 14. SWEEP3D degradation on Nehalem

Table 4. Percent of the active phase runtime that degrades application performance less than 5% for each PWM operation

Operation	Istanbul				Nehalem			
	SAGE	PARTISN	XNOBEL	SWEEP3D	SAGE	PARTISN	XNOBEL	SWEEP3D
Compute	100%	100%	100%	0%	100%	0%	100%	0%
L2 cache	75%	10%	75%	0%	25%	0%	100%	0%
L3 cache	10%	5%	10%	0%	5%	0%	25%	0%
Main memory	10%	5%	10%	0%	5%	0%	25%	0%

SWEEP3D on the Istanbul and Nehalem, as well as PARTISN on Nehalem. In contrast for all other applications the spare-core can be used 100% of the time for compute operations, between 25% and 100% of the time using private L2-cache, and between 5% and 25% of the time using either the shared L3 cache or the main memory. The exact impact is both application dependent and processor type dependent.

Note that a PWM cycle time of 1s was used to obtain all of the results above. The analysis was repeated for a cycle time of 5s and 10s which resulted in almost

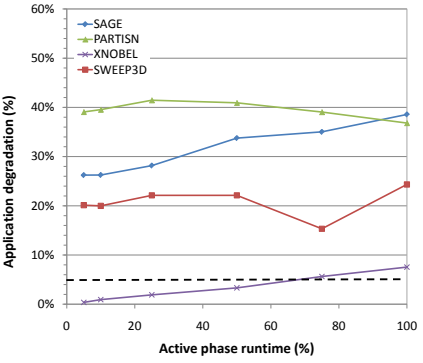

Fig. 15. Performance degradation for PWM network operations

identical results indicating that the impact on application performance of using a spare-core is invariant to the PWM cycle time.

Figure 15 shows the results of applying the PWM approach for network operations for the applications evaluated above on two nodes containing two 4-core AMD processors. As can be seen, the impact of network operations is application dependent where performance degradation reaches 38%, 37%, 7.5% and 24% for SAGE, PARTISN, XNOBEL, and SWEEP3D at 100% of the active phase runtime.

5 Related Work

There has been much prior work on the possible use of spare-cores. This has included using spare-cores to increase reliability, to increase performance, and to monitor processor activities. However, to our knowledge, no prior work exists that attempts to characterize the impact of using spare cores on the performance of scientific applications.

Work that aims to increase reliability includes [4,5] that examines the use of spare-cores to overcome faults that can arise in multi-core systems. Intermittent faults can arise due to manufacturing, thermal and voltage variations and are expected to become more common place in the future. Spare-cores could be used to overcome faults by migrating computation or vitalizing functionality when either transient or hard-faults occur.

The use of spare-cores for speculative execution and for data pre-fetching is also under exploration [6,7]. Such approaches have the promise of improving performance by using a spare-core to support a main core but do so at the cost of increased core-use and hence increased power use. Note that power-use is increasingly an important aspect to consider. The dedication of cores is also being explored for Operating-System activities to reduce performance variability caused by OS-Jitter or OS-Noise [8].

The use of spare-cores for monitoring activities while applications are running, such as sampling performance, thermal and power usage is also being explored, e.g. [9]. Typically these approaches use spare-cores independently of application cores and monitor activities which require occasionally storage of local buffers to main memory and ultimately to disk for post-processing. Our characterization of the impact of using spare-cores on application performance most closely resembles this latter case.

6 Conclusions

In this work we have analyzed the impact on application performance of using spare-cores for various classes of operations. We initially analyzed how applications utilize current state-of-the-art multi-core processing nodes from both AMD and Intel. Memory-bound applications are unable to take advantage of all cores in a processor, being bottlenecked by the main memory bandwidth, and thus leaving cores unused. In contrast compute-bound codes can utilize all cores resulting in increased performance and thus do not enable cores to be dedicated to other activities without a significant reduction in performance.

A pulse width modulation (PWM) approach was used to characterize the impact of using spare-cores. An active phase of each cycle is used to undertake an operation and an inactive phase leaves the core idle. By varying the active phase from between 0% and 100% of each cycle we explored the performance impact for various possible spare-core uses.

The results show that the performance impact when using spare-cores is both dependent on the application and on the actual processing nodes. By considering the scenario that a 5% performance impact was reasonable we showed that memory-bound applications enable significant computational tasks to be undertaken by the spare-cores as well as use of its private L2-cache. However, a spare-cores use of shared-L3 cache or main memory can significantly impact performance. In contrast compute-bound applications were significantly impacted by the use of cores for activities other than processing the application itself. As the number of cores per processor increases the impact to applications, even compute-bound applications, will be minimized.

The rapid growth in the available cores will lead to numerous opportunities for using spare-cores for support activities. We envisage that the PWM approach will be applicable to future multi-core processors allowing a characterization of spare-cores utilization to be achieved.

Acknowledgments

This work was funded in part by the Advanced Simulation and Computing program and the Office of Science of the Department of Energy. Los Alamos is operated by the Los Alamos National Security, LLC for the US Department of Energy under contract No. DE-AC52-06NA25396.

References

1. Intel: Futuristic Intel Chip Could Reshape How Computers are Built, Consumers Interact with Their PCs and Personal Devices (2009) Press released at, http://www.intel.com/pressroom/archive/releases/2009/20091202comp_sm.html
2. Barker, K., Davis, K., Hoisie, A., Kerbyson, D., Lang, M., Pakin, S., Sancho, J.: A Performance Evaluation of the Nehalem Quad-core Processor for Scientific Computing. Parallel Processing Letters 18(4), 453–469 (2008)
3. Sakuma, K., Andry, P.S., Tsang, C.K., Wright, S.L., Dang, B., Patel, C.S., Webb, B.C., Maria, J., Sprogis, E.J., Kang, S.K., Polastre, R.J., Horton, R.R., Knickerbocker, J.U.: 3D Chip-stacking Technology with Through-silicon Vias and Low-volume Lead-free Interconnections. IBM Journal of Research and Development 52(6), 611–622 (2008)
4. Wells, P.M., Chakraborty, K., Sohi, G.S.: Adapting to Intermittent Faults in Multicore Systems. In: Proc. ACM ASPLOS, Seattle, WA, pp. 255–264 (March 2008)
5. Joseph, R.: Exploring Salvage Techniques for Multi-core Architectures. In: Proc. Workshop on High Performance Computing Reliability in Conjunction with HPCA-11, San Francisco, CA (February 2005)
6. Zhou, H.: Dual-Core Execution: Building a Highly Scalable Single-Thread Instruction Window. In: International Conference on Parallel Architecture and Compilation Techniques, St. Louis, MO, pp. 231–242 (2005)
7. Ganusov, I., Burtscher, M.: Future Execution: A Hardware Prefetching Technique for Chip Multiprocessors. In: International Conference on Parallel Architecture and Compilation Techniques, St. Louis, MO, pp. 350–360 (September 2005)
8. Porterfield, A., Fowler, R., Neyer, M.: MAESTRO: Dynamic Runtime Power and Concurrency. In: Workshop on Managed Many-Core Systems Colocated with the ACM International Symposium on High Performance Distributed Computing, Boston, MA (June 2008)
9. Chow, J., Garfinkel, T., Chen, P.: Decoupling Dynamic Program Analysis from Execution in Virtual Environment. In: Proc. Usenix Annual Technical Conference, Boston, MA, pp. 1–14 (June 2008)
10. Kerbyson, D.J., Alme, H.J., Hoisie, A., Petrini, F., Wasserman, H.J., Gittings, M.: Predictive Performance and Scalability Modeling of a Large-Scale Application. In: Supercomputing Conference, Denver, Colorado, p. 39 (November 2001)
11. Koch, K.R., Baker, R.S., Alcouffe, R.E.: Solution of the First-order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor. Transactions of the American Nuclear Society 65, 192–198 (1992)