CPU-level resource allocation for optimal execution of multi-process physics code

Marta Bertran Ferrer[1], Costin Grigoras[1], Sergiu Weisz[2], Rosa M Badia[3]

CERN[1], University POLITEHNICA of Bucharest[2], Barcelona Supercomputing Center[3]

E-mail: marta.bertran.ferrer@cern.ch

Abstract.

The ALICE experiment data processing software relies on running parallel processes on multiple processor cores with large shared memory areas for data exchange. In the absence of mechanisms that guarantee job resource isolation, the deployment of these jobs can result in a usage that exceeds the resources originally requested and allocated. Internally, jobs may launch as many processes as defined in their workflow, significantly higher than the number of allocated CPU cores. This freedom of execution can be limited by mechanisms like *cgroups*, already employed by some Grid sites, currently a minority. If jobs are allowed to run unconstrained, they may interfere with each other in terms of the simultaneous utilization of the resources. Confinement mechanisms in this context improve the fairness of resource utilization, both between ALICE jobs and towards other users in general.

The efficient use of the cache memory of the worker nodes is closely related to the CPU cores executing the job. Important aspects to consider are the host architecture and the cache topology, i.e. cache levels, sizes and hierarchical connections to individual cores. Memory usage patterns of running tasks, the memory and cache topologies, and the CPU cores chosen to constrain the job influence the overall efficiency of the execution, in terms of useful work done per unit of time.

This paper presents an analysis of the impact of different CPU pinning strategies on the typical ALICE Monte Carlo job efficiency. The evaluation of the different configurations is performed by extracting a set of metrics related to job execution time. The results are presented both for the execution of an isolated job on and for whole node saturation. The effect of pinning is studied on different host architectures.

1 Introduction

The heterogeneity of the sites that comprise the ALICE Grid is significant, and so are the used job deployment resource allocation policies. In most of them, the worker nodes let jobs use their resources freely during their execution as they are not configured to apply any limitation or constraint. Some of the jobs running on the Grid do not confine themselves to using the resources they originally requested and to which they have been allocated, but exceed these limits by over-utilising the resources granted. This behaviour is unfair to other applications running concurrently, both from other users on the machine and other ALICE jobs.

One of the scheduling policies within Grid sites is the allocation of whole nodes for the execution of ALICE workflows. In this situation, the resource unfairness of the running payloads is of particular concern as the host batch queuing system is not involved in the job allocation process and cannot apply confinement mechanisms. In this case, the entire node is handled by JAliEn

[1], the ALICE Grid middleware, which is responsible for managing the node's resources and distributing them among the executing jobs. To be able to enforce resource utilisation fairness among jobs, a mechanism to constrain their execution to a given set of CPU resources has been implemented. An initial development was undertaken using CPU pinning techniques in which a given set of cores was allocated to the execution of individual jobs. Its implementation, together with a study of the configuration status of the different sites, is presented in Ref. [2]. In this first release, the cores to be allocated were given on a first-come first-serve basis, i.e. the available cores were assigned to the execution of jobs without following any kind of predetermined logic for their selection. For this assignment, the Linux command *taskset* [3] was used both to inspect the cores that had already been pinned by other executing workflows and to assign the determined set of cores to the launched job. However, the initial pinning workflow suffered from inefficiencies in the use of cache memories as the NUMA architecture of the systems was not taken into account to optimise the core selection.

Figure 1 shows the CPU consumption of a 32-core idle machine when running an over-consuming eight-core job. The running job uses 50% of the machine's resources, i.e. 16 CPU cores, twice as much as its original allocation. The CPU consumption when the same job is explicitly pinned to a set of eight cores using the described methodology is shown in Figure 2.

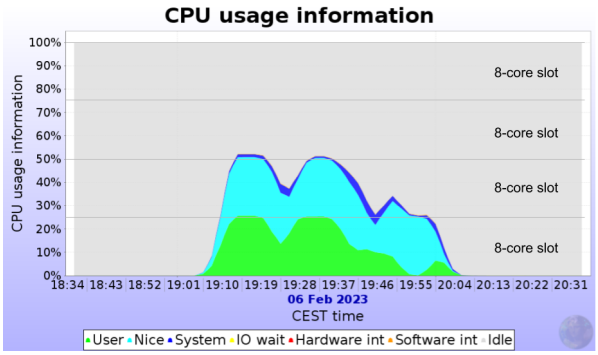

Figure 1: CPU usage of over-consuming eight-core job running freely on the machine.

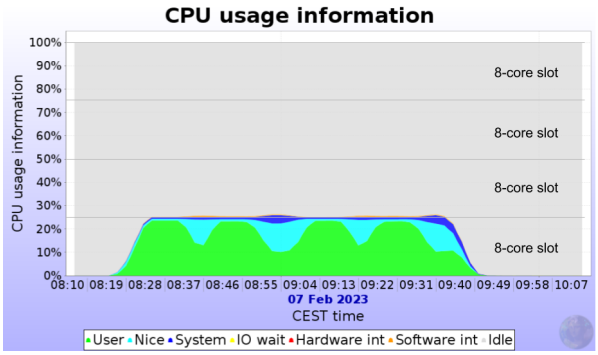

Figure 2: CPU usage of over-consuming eight-core job constrained to eight cores.

These proceedings present an analysis of the impact of executing physics simulation multi-core tasks in different CPU pinning configurations to observe their impact on the overall job efficiency and execution time. Section 2 gives an overview of some of the approaches used for executing non-interfering concurrent workloads and for optimising data locality on NUMA architectures. Section 3 presents the different CPU core selection methodologies evaluated. Lastly, Section 4 describes how the candidate pinning configurations are applied to scenarios in which a single job was being executed on an idle machine and also in which all the node's CPU resources were being saturated with multi-core tasks.

2 State of the Art

2.1 Workload co-location techniques

Workload colocation techniques are a must for exploiting the full potential of multicore architectures. A likely pitfall in their application is the workload interference on the shared resources, where coinciding peak use can lead to anomalies [4]. To overcome such a scenario, one can use a technique of workflow isolation by restricting it to a specific subset of resources. In the case of CPU resources, this is known as CPU pinning and its effects on a running task are closely related to the workload type and the physical system architecture [5].

The system architecture defines the structure of cache memories and how they are shared between CPU cores. From this configuration and the specific resources usage patterns of a task, the most appropriate subset of CPU cores can be selected for its execution [4]. Techniques for mapping threads to cores lead to reduced cache misses, improved memory access, and reduced

traffic on slower and power-expensive inter-chip connections. Given the potential that CPU pinning can have in eliminating interference, it is interesting to study various configurations in different scenarios taking into account both the architecture of the machine, the degree of CPU resource utilization, and the behavior and nature of the workflows.

2.2 Memory access

Workload memory access optimization is a major factor for good levels of execution efficiency. As today's applications have high data volume demands, memory systems have evolved towards deep and complex hierarchical models that minimize processor access latency levels [5]. Current hierarchies, known as Non-Uniform Memory Access (NUMA), consist of cache memories distributed in private and shared tiers between processors orchestrated by memory controllers, the data movements between which penalize performance and energy consumption [6]. The architecture of the executing machines is based on a set of NUMA nodes each maintaining its own memory controller. The processor cores are connected to the different levels of the cache memory depending on the CPU architecture.

Memory access by cores can be orchestrated by a local or remote controller, and this can have an impact on both power and memory consumption, with local accesses being more efficient [6]. The usage levels of the different memories are another factor that can also affect efficiency, possibly causing a reduction in efficiency due to cache misses. The memory access performance has special significance in multi-core and multi-threaded architectures as they have to supply large quantities of data to the many concurrent functional units. To optimize the data accessed by the running threads, these can be assigned to cores in a way that is optimal for the cache levels and their connections. As there is no traversal of memory interconnect, intra-chip traffic is favored over the inter-chip traffic, which has a higher latency and lower bandwidth. In applications where inter-process communication is performed, placing communicating threads near each other improves speed through the use of shared caches.

This is particularly relevant for ALICE, as the software stack is built around a message-passing framework with a memory-sharing large number of process [7]. Another factor that penalizes the execution of concurrent tasks is the shared cache contention [8]. Applications may have phases with access overlap, which is prone to cause significant variability in the observed performance levels. The latter are largely determined by how the memory accesses are interleaved between concurrent applications.

3 CPU pinning configurations study

In order to evaluate the impact of different CPU core allocation techniques, a set of selection methods are proposed to evaluate the effects of sharing or spreading cache access and the use of interconnecting links. Five specific configurations are evaluated to assess the containment of jobs as well as the selection of a specific set of cores, including the effects of using shared resources with the cooperating cores.

- **Configuration 1**: Same NUMA Node and independent Level 1 + Level 2 (L1+L2) cache.
- **Configuration 2**: Different NUMA Nodes and independent L1+L2 cache.
- **Configuration 3**: Same NUMA Node and sharing L1+L2 cache.
- **Configuration 4**: Random core assignment.
- **Configuration 5**: No pinning performed. Jobs run freely on the machine without any limitation being applied.

For our studies, multi-core simulation jobs implemented on top of the O2 framework [7] were executed. These jobs are characterised by the internal deployment of parallel processes that perform inter-process communication using shared memory. To enhance the full utilisation of the available resources, they launch a surplus of processes with a lower priority that makes them prone to exceeding the allocated bounds in the absence of external limitation mechanisms. These

jobs were run in different scenarios, both on idle and saturated machines, as well as assigned to different sets of cores following the configurations listed above.

To evaluate the different configurations, the performance parameters of the jobs and their usage of CPU resources were monitored. Specifically, a periodic parsing of the */proc/stat* file was used to account for the time spent on each of the CPU components. In addition, the job execution time was tracked from the output of the *time* command, being subdivided into *real* (total execution time), *user* (time spent in user mode) and *sys* (time spent in kernel mode).

4 Results

4.1 Experimenting with one multi-core job running on idle machines

The first case study was the execution of a single eight-core job on idle machines with AMD and Intel CPUs. In both cases, among the four pinning configurations the shortest execution time was observed in Configuration 1 - Same NUMA Node and independent L1 and L2 cache. In the configuration where no pinning was applied, Configuration 5, a CPU consumption that went above the originally requested eight cores was observed given the jobs' propensity to deploy extra, lower-priority processes that can spawn on unused cores. The fact that no limitations were imposed and that more cores than originally allocated were used made these jobs on average complete in 10% less (*real*) time among the test machines. However, the *user* time and *system* time were higher than in Configuration 1. Particularly significant is the reduction in *system* time in which a pronounced decrease of 30% on average was observed, indicating that kernel mode operations could be completed in a shorter time interval given the reduction in the use of inter-chip connections and a more efficient data locality. Figures 1 and 2 illustrate the case of one of the test machines where the increase in execution time is significant (45%), although the integral of CPU time across all components is reduced by applying the CPU constraint. Nevertheless, this is not a realistic situation as in production all machine resources usually are saturated and used for the execution of multiple jobs in parallel.

4.2 Saturating machines with multi-core jobs

A comparison of the execution times of an idle machine versus a saturated one reveals the effects of hyperthreading on saturation, which leads to an overall increase in job execution times. On idle machines the Turbo Boost / Turbo Core functions of the CPUs are enabled, thereby improving their performance.

In this case, the best results within the described pinning configurations are also obtained when applying Configuration 1 to the CPU core selection. The jobs are particularly benefited in the usage of the CPU's *user* and *system* components. In Configuration 5, executing jobs freely compete for the resources of the machine, which results in the natural allocation of a portion of the resources proportional to the number of running tasks. However, over the course of their execution, jobs may suffer from slight desynchronisations that cause their peak resource utilisation to fluctuate over time. This causes idle portions of resources to become available for use by other workflows, so that during part of their execution they may make use of more resources than originally allocated. These peaks in resource usage result in the job being able to complete in a shorter wall-clock time. However, given that resources are not used in the most optimal way to exploit the system's architecture, operations in the user and kernel modes are more expensive and therefore take longer to complete.

Job desynchronisation has a significant impact on job execution time and its variability. Pronounced desynchronisations are observed on machines where jobs are tied to cores of varying performance levels. It is observed that on some machines the performance of the cores is strongly impacted by factors like the location of the PCI ports in the NUMA architecture. The behaviour of the machine's CPU cores is not uniform, and is directly correlated to its physical architecture.

Other factors that have a direct influence on the job execution duration are the CPU frequency and the size of the Random Access Memory (RAM). The higher the CPU frequency and memory size (up to the usable limit), the faster the jobs are executed.

Figure 3 shows the combined time measurements of two of the target execution machines when saturating all their cores with eight-core simulation jobs. The results are the averages of user and respectively system CPU time of all executions across the two platforms. A total of 60 parallel multi-core jobs were run in each of the configurations. Time measurements are shown for the best performing pinning configuration — Configuration 1 — against the non-pinned one — Configuration 5. The user and system times of the other configurations (Configurations 2, 3 and 4) have values ranging between these two. Both are Intel machines that differ in the number and model of cores:

- 2 x 16 HT CPU cores - Intel Xeon Gold 6244 @3.6GHz
- 2 x 32 HT CPU cores - Intel Xeon Gold 6226R @2.90GHz

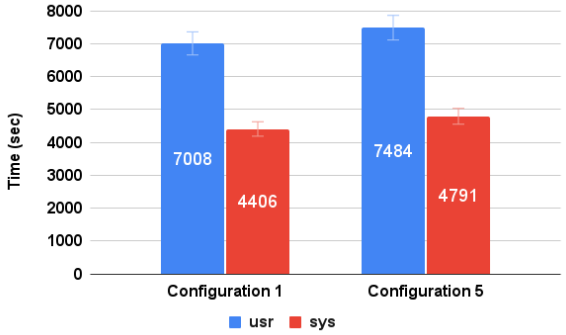

Figure 3: Combined *user* and *system* time values of jobs running in Configuration 1 and Configuration 5 (unpinned).

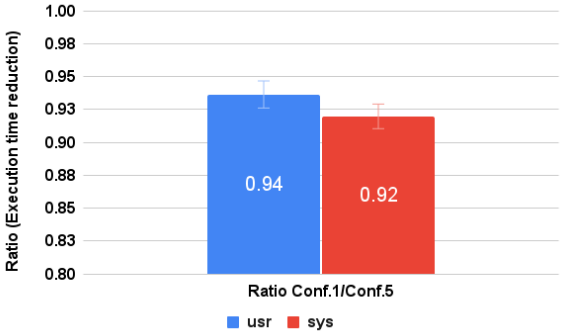

Figure 4: Ratio for *user* and *system* components of jobs executed in Configuration 1 over Configuration 5 (unpinned).

The measurements obtained from running repeated jobs reveal how in both cases the results are compatible between the two systems. Figure 4 presents the ratio of the time measurements obtained in both configurations. A decrease in *user* and *system* times is clearly seen, which indicates that the jobs were able to complete their execution using the resources for a shorter time. This reveals resources were used more efficiently, taking advantage of improved data locality and intra-chip connections.

4.3 Implementing dynamic CPU confinement

The large variability of the resource usage patterns across the set of jobs running on the Grid leads to unpredictable and potentially very high CPU utilisation that exceeds the resources originally allocated.

On the one hand constraining the jobs to the requested amount of resources would lead to predictable execution times and fairness towards other users of the shared resources. On the other hand it would not exploit the potential of idle times on the node. As a compromise between the two the execution in those cases might only be constrained in the cases where the payloads consistently overuse the amount of allocated resources.

CPU resource utilisation levels are continuously tracked by the Application Monitoring service (*ApMon*) of the MonALISA framework [9], which runs on all Grid worker nodes. When a higher usage than the original allocation is detected, a grace period is established in which the jobs are allowed to use 20% more of the requested CPU portion. As shown in Fig. 5, when this over-consumption is maintained for a certain period of time, the job is restricted to the requested cores.

One of the features of *taskset* that made us decide to use it as a resource-limiting tool is that it can be used on already running workflows. For the dynamic confinement of the jobs, it is explicitly applied to each of the running processes and then inherited by their executing threads.

From this moment on, future spawned processes and threads will implicitly inherit the applied set of cores from their parent processes.

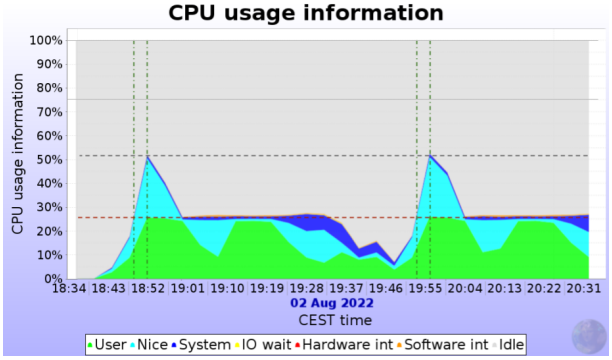

Figure 5: Dynamic constraining of cores upon detection of CPU over-utilisation. Horizontal lines show the CPU usage levels of a 32-core machine. Vertical lines delimit the grace period they are given to run unrestricted before being permanently constrained.

5 Conclusion

These proceedings detail the methodology used to constrain the execution of jobs to a set of CPU cores using different selection algorithms. The effects of CPU pinning in different environments — idle and saturated machines — were studied and their impact on job execution performance was analysed. CPU resource selection patterns were identified that optimise the efficiency of multi-core jobs as a function of the hardware architecture of the executing system. Compute servers of different manufacturers and architectures were employed to perform this study, thus validating a uniform and homogeneous behavior.

In the long term, it is expected that the predominant situation on the Grid worker nodes will be node saturation with multi-core jobs. This scenario was simulated in the case studies and significant benefits in job turnaround and CPU resource consumption were identified. Though the use of the new ALICE data processing software is still in its initial stages, a tendency of the jobs to consume more resources than originally allocated was identified. As a first step we have deployed the rigid constraining of payloads to the requested number of cores on Grid sites. Following that dynamic constraining is intended to be rolled out, tuning the grace time and overuse thresholds based on the experience and monitoring data collected from the first deployments.

6 Bibliography

[1] M Martinez Pedreira, C Grigoras, and V Yurchenko. JAliEn: The new ALICE high-performance and high-scalability Grid framework. In *EPJ Web of Conferences*, volume 214, page 03037. EDP Sciences, 2019.

[2] S Weisz and M Bertran Ferrer. Adding multi-core support to the alice grid middleware. In *Journal of Physics: Conference Series*, volume 2438, page 012009. IOP Publishing, 2023.

[3] taskset(1) — Linux manual page. `https://man7.org/linux/man-pages/man1/taskset.1.html`.

[4] A Podzimek, L Bulej, LY Chen, W Binder, and P Tuma. Analyzing the impact of cpu pinning and partial cpu loads on performance and energy efficiency. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 1–10. IEEE, 2015.

[5] M Diener, EHM Cruz, MAZ Alves, POA Navaux, and I Koren. Affinity-based thread and data mapping in shared memory systems. *ACM Computing Surveys (CSUR)*, 49(4):1–38, 2016.

[6] C Lameter. Numa (non-uniform memory access): An overview: Numa becomes more common because memory controllers get close to execution units on microprocessors. *Queue*, 11(7):40–51, 2013.

[7] P Buncic, M Krzewicki, and P Vande Vyvre. Technical Design Report for the Upgrade of the Online-Offline Computing System. CERN-LHCC-2015-006, ALICE-TDR-019. Technical report, 4 2015.

[8] A Sandberg, A Sembrant, E Hagersten, and D Black-Schaffer. Modeling performance variation due to cache sharing. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 155–166. IEEE, 2013.

[9] C Grigoras, R Voicu, N Tapus, I Legrand, F Carminati, and L Betev. Monalisa-based grid monitoring and control. *The European Physical Journal Plus*, 126(1):1–7, 2011.