Detection, Evaluation and Mitigation of Resource Affinity and Communication Contention Problems in a Task-Based Runtime over Heterogeneous Clusters

Lucas Leandro Nesi, Lucas Mello Schnorr

Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{llnesi,schnorr}@inf.ufrgs.br

Abstract. *The complexity of high performance computing (HPC) platforms presents challenges in parallel application development. The Task-Based paradigm is a candidate to reduce some of the programmer's burden. However, because of the platforms' complexity, resource affinity and communication contention might cause performance problems. This work presents a case study of these problems employing the Chameleon dense algebra linear solver LU factorization using the Task-Based runtime StarPU over 21 heterogeneous nodes. We present possible configurations to mitigate performance degradation and conduct an extensive analysis of their interaction. The results show a performance improvement of 16% without changing the application source code.*

1. Introduction

The evolution of HPC platforms increases the development's complexity of parallel applications [Dongarra et al. 2017]. Robust multi-core NUMA architectures, devices heterogeneity with accelerators like GPUs, and multiple nodes for distributed executions are examples of such challenges. The Task-based programming paradigm aids in developing such applications in these environments because they transfer some responsibilities to a dynamic runtime. Task-based applications use a more declarative way of programming that permits many dynamic decisions [Augonnet et al. 2011]. These runtimes schedule tasks to workers associated to a processing unit. Configuring such runtimes to adapt to such platforms is desirable and challenging due to the vast configuration space.

The complexity of such platforms, with NUMA and Heterogeneity, combined with runtime configurations, influences application performance. Moreover, the detection of problems and their causes is difficult. Applications' execution traces with visualizations may help to detect and examine negative behavior in such platforms. Moreover, even if a problem is detected, it may be challenging to find solutions. While application changes may be inevitable to mitigate these problems, sometimes the runtime settings are sufficient to solve these problems. The challenge is to evaluate the vast number of configurations possibilities and interactions among application, runtime, and platform. StarPU [Augonnet et al. 2011] is an example of task-based runtime with multiple necessary configurations to adequate itself to different types of computing platforms.

This work's motivation comes from the execution of the LU task-based factorization from the Chameleon dense algebra linear solver [Agullo et al. 2010] on top of a heterogeneous multi-node platform. The execution traces show some evident performance problems, idle times, and outliers on some nodes. This work uses this case study

as motivation to investigate which StarPU configurations enable problem mitigation on these sophisticated platforms. The main contributions are as follows. (i) The presentation of the study case and the methodology used to identify NUMA affinity and contention problems on the Task-based Chameleon LU operation. (ii) The analysis and enumeration of possible StarPU and a Linux kernel configuration that might mitigate the problem. (iii) An extensive statistical analysis considering all configurations and their interaction among each other, revealing a performance gain of 16%. All data of this work is publicly available online in a reproducible companion [1].

Section 2 presents the study case using the LU factorization of the Task-Based Chameleon dense algebra linear solver over 21 heterogeneous nodes. Section 3 enumerates the possible StarPU and Linux kernel configurations to mitigate the problem, with all the experimental methodology. Section 4 presents the experimental evaluation and analysis of all factors and their interactions. Section 5 presents other works that evaluate related problems both in HPC and, more specifically, in task-based runtimes. Section 6 concludes this work with the main observations and future work.

2. Study-Case: Chameleon on 21 nodes

This work's motivation comes from a representative execution of the LU factorization from the Chameleon Solver [Agullo et al. 2010] using two different clusters on Grid5000 platform [Bolze et al. 2006]. Chameleon is a dense linear algebra solver that uses task-based runtimes, including StarPU, to deal with computational platforms. The execution used 21 nodes, 14 from the `chetemi` cluster and seven from the `chifflet` one. The `chetemi` cluster has two Intel Xeon E5-2630 v4 with 10 cores per CPU, while `chifflet` cluster has two Intel Xeon E5-2680 v4 with 14 cores per CPU and two GPUs NVIDIA GeForce 1080Ti. The interconnection of the machines is a 10Gbps Ethernet. The operational system of the machines is Debian 10. The StarPU version is the April `nmad` optimized branch[2] [Denis et al. 2020] using the NewMadeleine suite [Aumage et al. 2007] MPI implementation. We used a modified Chameleon 0.9.2 to accept our custom data node distributions to consider node heterogeneity. This execution uses the default configurations except for removing two CPU workers for the MPI and main application thread, and the `DMDAS` scheduler. The LU factorization workload is a matrix of size 96000x96000 divided into 100×100 blocks of size 960x960.

Figure 1 presents the visualization of the described execution using the StarVZ Framework [Pinto et al. 2018] [Nesi et al. 2019]. The visualization is the Node Occupation panel, which depicts the aggregated node resource type (CPU/GPU) utilization per time steps of 100ms. The height of the bar presents the utilization of the same type of resources per timestep per task (the bar's color). The left-side number, 48099, is the total makespan of the execution in milliseconds. Nodes 0 to 13 are the `chetemi` cluster, and only contain one line because they only have CPUs, while the nodes 14 to 20 are from the `chifflet` cluster and have a line for the CPUs and one for GPUs. The main interpretation is that there are idle times throughout the execution (white areas), especially on the `chifflet` nodes. These idle times were caused by delayed task dependencies executed in other nodes. To understand the reasons behind this, we select one node from the `chifflet` to make an individual visualization of the node's workers.

[1] `https://gitlab.com/lnesi/wscad2020-companion`

[2] Branch: `nmad-coop-coll-dynamic-interface` with `STARPU_MPI_COOP_SENDS=1`

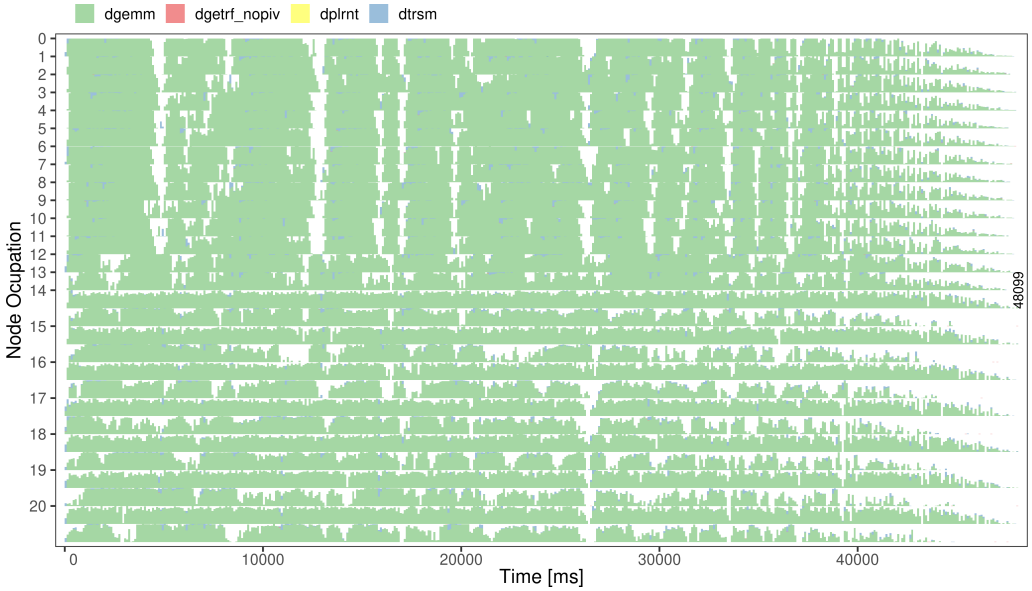

Figure 1. Aggregated node visualization for the LU factorization over 21 nodes.

We select node 20, from the `chifflet` cluster, to present its workers' visualization, which is a good example of the general idle behavior. Note that this execution with 21 nodes haves 434 workers, so a full visualization is impracticable to be shown here. Figure 2 (Left) shows the Application workers panel, depicting a Gantt chart of tasks for the node 20 workers (Y-axis). Each task is a state where colors represent the task type. The visualization divides tasks into two groups, non-outliers, that have an opacity of 50% (lighter colors), and outliers, that do not have opacity (strong colors). The Application workers panel shows that the majority of `dgemm` tasks of resources CPU12 to CPU23 are outliers. Notably, these resources are CPUs from NUMA node 1. Moreover, there are some tasks of GPU CUDA0_0 attached to NUMA node 0 that are outliers. For these visualizations, an outlier is a task that has a duration superior to 1.5 times the inter-quartile range from the third quartile. The duration of these outlier tasks can be very different from the normal tasks. Figure 2 (Right) presents boxplots for the durations of `dgemm` tasks (X-axis) for each worker (Y-axis) of node 20. The duration of the tasks on NUMA node 1 is higher, as shown by the mean and quartile values, and more unstable, as the interquartile range is larger. Also, CUDA0_0 presents slower outliers than CUDA0_1. These observations indicate that problems of data/thread affinity and contention could be occurring in this execution, and strategies or configurations to mitigate it are desirable.

3. Design of Experiments

This section presents the configurations and the design of experiments to mitigate the stated problem. The decisions about thread and data mapping naturally need to consider the machine's architecture. Figure 3 presents the structure of one `chifflet` node produced by `hwloc` [3]. We have removed from this view all unused devices connected to NUMA 0, such as disks, unused PCIs, eno3, and eno4. The machine has two NUMA nodes. Each NUMA node has 378GB of RAM and 14 physical cores. All network interfaces are in NUMA node 0. Moreover, this machine has two GPUs, one per NUMA node.

[3] `https://www.open-mpi.org/projects/hwloc/`

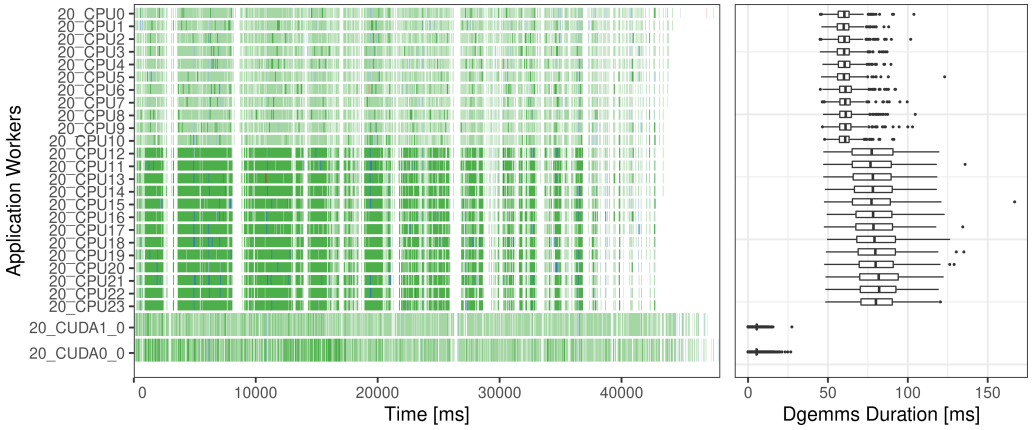

Figure 2. StarVZ space-time visualization for Node 20 (Right) and Duration of dgemms tasks per worker (Left).

This information guides our decisions about future configurations.

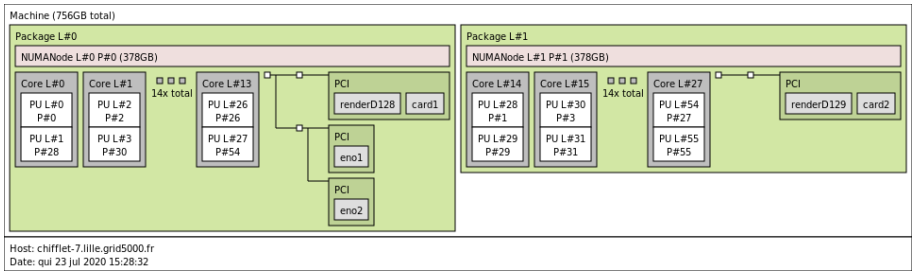

Figure 3. Hardware topology for the Chifflet machine generated by hwloc.

Instead of developing or using external strategies to mitigate the problem, this work focus on using already present configurations of the software stack, mainly from StarPU but also from the Linux kernel. The internal structure of StarPU consists of a worker per computing resource. Each worker is a CPU thread. Accelerators also use a dedicated CPU thread. Moreover, StarPU can dedicate a CPU thread for the MPI manager, and the main application thread (that submits tasks). The binding of these CPU threads considering the entities that they use could influence the performance of tasks. We investigate StarPU documentation and select possible settings of interest that may impact this situation. We enumerate seven experimental factors that have different objectives for the problem mitigation and list the parameters used to achieve them. All factors have only two levels and consider only if the configuration is enabled (assuming value 1) or disabled (Default, assuming value -1). The list of factors is as follows.

Factor A. ***Objective***: Split memory RAM between NUMA nodes to maximize data affinity on CPU workers. StarPU will create a memory manager per NUMA node and move data to the respective worker's memory manager before running the tasks. This communication is concurrent with other tasks' computation. ***Configuration***: StarPU environment `STARPU_USE_NUMA`. ***Default***: One memory management for the RAM, where the OS will control application memory allocation, probably on NUMA node 0. ***Enabled Value***: One memory management per NUMA node.

Factor B. ***Objective***: Place each CPU thread managing a GPU on the same NUMA

node of the device. ***Configuration***: StarPU environments `STARPU_WORKERS_CPUID` and `STARPU_WORKERS_CUDAID`. ***Default***: StarPU places the GPU worker's CPU thread on the first two possible CPU workers IDs. Usually, on NUMA node 0. ***Enabled Value***: Place each GPU worker CPU thread on their respective NUMA node.

Factor C. ***Objective***: Place the MPI Thread on the NUMA node of the network adapter. ***Configuration***: StarPU environment `STARPU_MPI_THREAD_CPUID`. ***Default***: If a CPU worker is reserved, StarPU will use the last CPU worker, usually in NUMA node 1. ***Enabled Value***: Set the MPI thread on a NUMA node 0.

Factor D. ***Objetive***: Move application memory and main thread to NUMA node 1. StarPU will allocate memory for the application on the first CPU worker's memory manager, usually NUMA node 0. The NUMA node 0 may be overloaded with so many devices, and moving the application memory and application main thread to a different NUMA node could help. ***Configuration***: StarPU environments `STARPU_MAIN_THREAD_CPUID` and `STARPU_WORKERS_CPUID`. ***Default***: Usually, the first CPU worker will be on NUMA node 0, used as the disabled value. ***Enabled Value***: Place application main thread and memory on NUMA node 1.

Factor E. ***Objective***: Some schedulers consider data transfer durations when scheduling. StarPU computes the estimated time for transferring data by a one-time calibration (if not already calibrated) of the bandwidth between all memory managers. However, StarPU is optimistic because the PCI bus will be fully available for each pair's calibration. On the real execution, memory, GPUs, and network will use the PCI bus at the same time. Some schedulers, the case of `DMDAS`, use the BETA coefficient to multiplies the calibration time when computing this transfer time. ***Configuration***: StarPU environment `STARPU_SCHED_BETA`. ***Default***: one, the scheduler uses the original calibration value. ***Enabled value***: 10, make the data movement ten times slower for scheduling purposes, biasing StarPU to avoid data transfers.

Factor F. ***Objective***: Increase the number of asynchronous tasks submitted to GPUs. This setting may affect the number of GPU's outlier tasks. ***Configuration***: StarPU environment `STARPU_CUDA_PIPELINE`. ***Default***: Two tasks. ***Enabled value***: Four tasks, maximum value permitted by this configuration.

Factor G. ***Objective***: The Linux kernel has a feature to automatic balance memory between NUMA nodes by the number of page faults and how often the nodes access it. Because StarPU should be controlling all the memory, the kernel may be impairing the performance. This may be related to Factors A, C and D. ***Configuration***: Kernel configuration `kernel.numa_balancing`. ***Default***: Enable. ***Enabled value***: Disabled.

After some preliminary tests, we observed high interaction between factors and decided for a full factorial experimental design [Jain 1990]. It consists of these seven factors with two levels and 30 repetitions totalizing 128 different configurations and 3840 randomized experiments. All the settings are equal to the Section 2 study case.

4. Experimental Evaluation

With the 3840 measurements, we fit a linear model ($Time \sim A * B * C * D * E * F * G$), where each factor can assume value -1 for configuration disabled and 1 for configuration enabled [Jain 1990]. The model has a Multiple R-squared of 0.9825. Moreover, we con-

duct an analysis of variance (ANOVA). Table 1 shows the summary of results for the most prominent terms (absolute estimations $>= 0.1$, P-Values $< 1e-30$), with the coefficients estimations, the estimations lower (LC) and upper (UC) 95% confidence intervals, the ANOVA Sum square, F-Value, and P-Value in conjunction with the percentage of variance explained by the term. The first line contains the intercept, and the last the residuals.

Table 1. Linear Model estimations for factors and analysis of variance.

Term	Estimate	LC	UC	Sum Sq	F-value	P-value	% Exp
(Intercept)	45.234	45.219	45.25				100
A	-3.349	-3.364	-3.334	4.3e+04	1.8e+05	0.0e+00	86.193
A:E	-0.675	-0.691	-0.660	1.8e+03	7.4e+03	0.0e+00	3.505
E	-0.481	-0.496	-0.465	8.9e+02	3.8e+03	0.0e+00	1.775
B:D	0.385	0.370	0.400	5.7e+02	2.4e+03	0.0e+00	1.140
A:D	-0.325	-0.341	-0.310	4.1e+02	1.7e+03	3.4e-310	0.813
A:B:D	0.310	0.295	0.326	3.7e+02	1.6e+03	1.4e-286	0.739
C	-0.292	-0.307	-0.277	3.3e+02	1.4e+03	1.0e-258	0.655
A:B:D:E	-0.266	-0.281	-0.250	2.7e+02	1.2e+03	2.6e-220	0.543
B:D:E	-0.258	-0.273	-0.243	2.6e+02	1.1e+03	3.8e-209	0.512
C:D	0.254	0.238	0.269	2.5e+02	1.0e+03	7.7e-203	0.494
A:C:D:E	-0.251	-0.267	-0.236	2.4e+02	1.0e+03	1.1e-199	0.485
C:D:E	0.172	0.157	0.188	1.1e+02	4.8e+02	7.5e-101	0.228
G	-0.169	-0.185	-0.154	1.1e+02	4.7e+02	9.6e-98	0.220
B	0.128	0.112	0.143	6.3e+01	2.7e+02	9.2e-58	0.125
A:B	-0.119	-0.135	-0.104	5.5e+01	2.3e+02	5.0e-51	0.110
B:C	0.119	0.104	0.135	5.5e+01	2.3e+02	6.7e-51	0.109
A:C	0.119	0.103	0.134	5.4e+01	2.3e+02	2.6e-50	0.108
D	0.112	0.097	0.127	4.8e+01	2.0e+02	3.5e-45	0.096
Residuals				8.7e+02			1.749

The Table 1 data show that Factor A is the most prominent coefficient (-3.3s) and accounts for most of the variation (86.1%), following by the interaction of A and E (-0.6s and 3.5%) and Factor E alone (-0.4s and 1.7%). This situation clearly shows that the best case is with Factor A enabled. However, this analysis accounts for variations of some bad situations that are not of interest, and the objective is to minimize the makespan. It may be the case that some factors present some variations or behaviors only when Factor A is disabled. Instead of using this data to make all decisions, we will break the situations (Use or not of the factors) refining the conclusions.

Figure 4 presents the main effects plot of all factors. We analyze the influence of each factor (on the X-axis: disable as in -1, or enable as in 1) in the execution time (the Y-axis). As already discussed, factors A and E have the greatest impact on execution time. Moreover, Factors B and D seem to cause little negative effects. Because we suspect that this behavior is related to cases where Factor A is disabled, the inspection of Factors interactions is necessary. Because of the magnitude of estimations of Factors impact, we break it into two Figures.

Figure 5 shows the interactions of Factor A with all others, while Figure 6 shows the interactions among the rest of the factors. These interaction plots present a facet for each pair of factors with their interaction, containing the four possible configurations. Each factor is enabled (1) or disabled (-1) for each configuration. The vertical factor

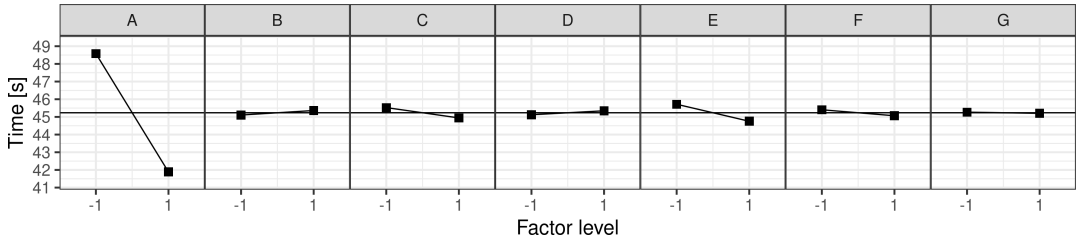

Figure 4. Main effects plot for all factors.

(column) has its value changing from -1 to 1 in the X-axis, while the horizontal factor (line) has its value split into two lines (red for -1 and black for 1). The Y-axis presents the makespan for the configuration. The red-left point in each facet represents both factors disabled while the black-right point represents both enabled.

The interactions of Factors A/E present in Figure 5 annotation 1 shows that E=1 indeed improve execution time when A=1, this was also checked in the ANOVA table. These situations suggest that E, following A, is a strong candidate to be always enabled. Because Factor A is predominant, the Y-axis of Figures 5 and 6 are different. Figures 6 shows that interactions of B/D and C/D can have negative impacts (execution's makespan increase). If B or C are enabled, and D is activated, there will be a negative impact, as shown in annotation 2. As well, if D is enabled, and B is also enabled, the makespan will increase (Annotation 3). Factor E was a good candidate to be always enabled, checking interference with other factors may be interesting. The interaction of D/E on annotation 4 revels that if E is enabled, and D becomes enabled, there will be a drop in performance. Also, if E is disabled, a very small drop in performance exist. These indicate that D has a negative effect on performance when Factors B, C, and E are enabled.

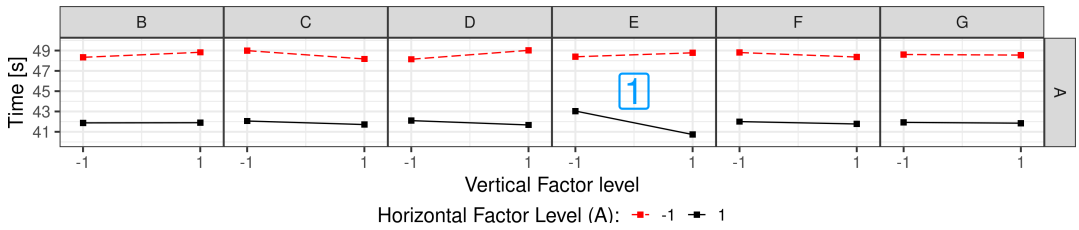

Figure 5. Interation of Factor A with all other factors.

Figure 7 (Left) present box plots of the execution times when varying the values for Factors A and E. It is important to state that it contains data with all possible factor levels. It is possible to observe that times are worst when A is disabled. However, the combination with A=1 and E=-1 still has some higher maximum values that intercept with the minimum values of both A=-1 cases. Also, when combining A=1 with E=1, the mean and quartiles became lower and stable. Still, there are many outliers but could be of a specific case of other factors interaction. With these observations, A and E enabled presents advantages. The remaining analysis will always consider A=1 and E=1 to discover the impact of other factors. Figure 7 (Right) present the main effects of the remaining Factors. In this case, all factors have positive impacts (makespan drops), with B, C, and D having higher impacts. This result is different from the earlier main effects plot, showing the harmful effects of the factors in situations where A=-1 or E=-1.

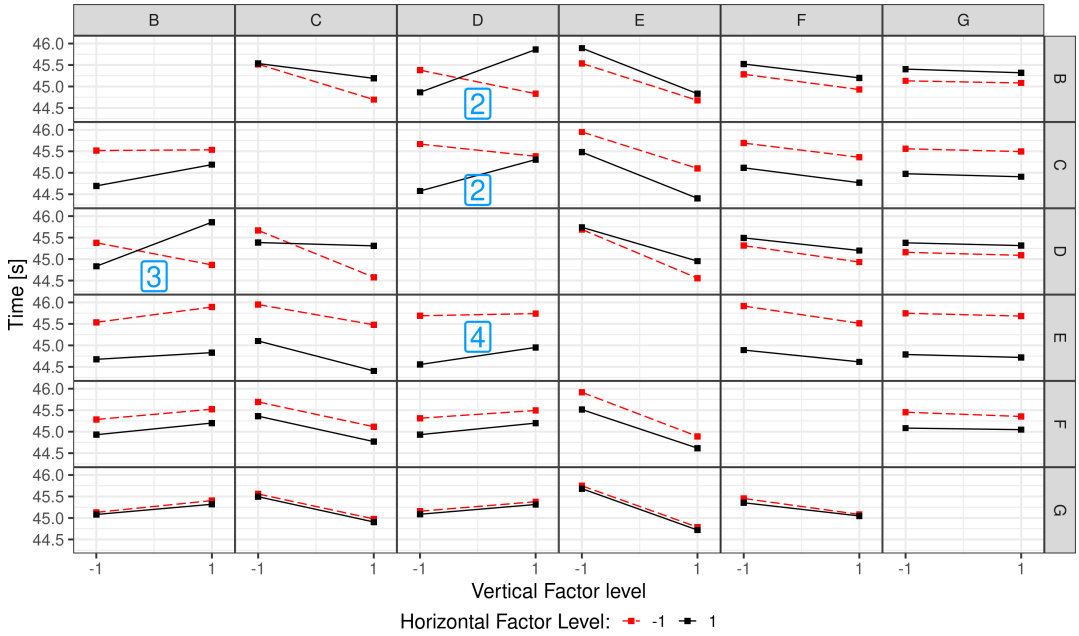

Figure 6. Interation among Factors B, C, D, E, F, and G.

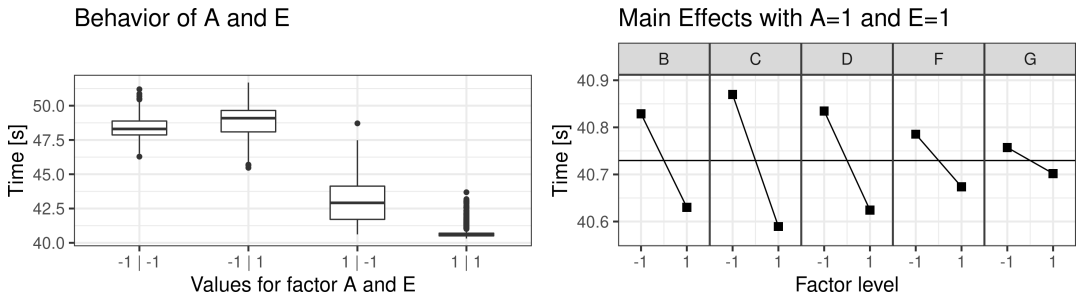

Figure 7. Boxplot for different values of Factors A and E (Left) and Main Effects with A=1 and E=1 for the remaning Factors (Right).

The analysis of the interaction between the remaining Factors could reveal the potential candidates for enabled settings. Figure 8 presents it. First, in annotation 5, it is possible to verify that B and C do not have almost any interaction when at least one of them is enabled, and a huge impact from both disabled to one enabled. However, interactions of D with B or C are prejudicial (Annotation 6) if at least one of the Factors is enabled. Because B, C, and D are the most prominent factors, and there is this negative interference, we believe that the choice would be enabling B and C, or enabling D.

Figure 9 (Left) presents a boxplot for the different values of B, C, and D with A and E enabled, representing all nine possibilities. The first thing observed is that in the case of all three disabled, the makespan is the worst situation, and the variance is very high. Activating any of the three factors causes a very positive impact. The case with only C has many outliers, and the case with B only has a larger mean compared to just D. Activating both B and D causes the appearance of many outliers, and all three enabled factors have this same problem. This situation reinforces the observations of the interaction plot. Case B and C presents the lowest mean and very stable measurements; however, it is very close to the only D case. Also, it is important to notice the Y-axis

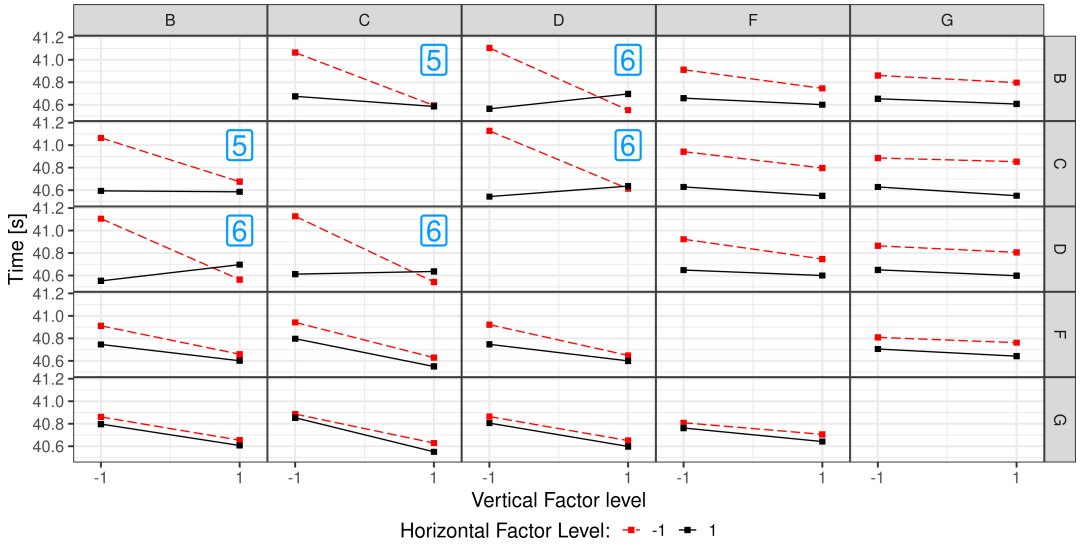

Figure 8. Interation plot with A=1 and E=1 for the remaning Factors.

values. The gains are not expressive. In this case, analyzing what the factors mean could be the answer for this case. Factor C represents that StarPU will place the MPI worker thread at the same NUMA node as the ethernet adapter, NUMA node 0, so this seems a pretty logical thing to do, and disabling C does not harm the execution. Factor D states that StarPU will place application thread and main memory at NUMA node 1. If D and C are enabled, many outliers appear, stating that the application's main thread and memory should be at the same NUMA node of the MPI thread. Because of this, we consider the best case to be using C and not enabling D, thought, only enabling D would have similar results. Because enabling B with C only causes positive results, we consider, until now, the configuration A=1, B=1, C=1, D=-1, E=1 the best situation.

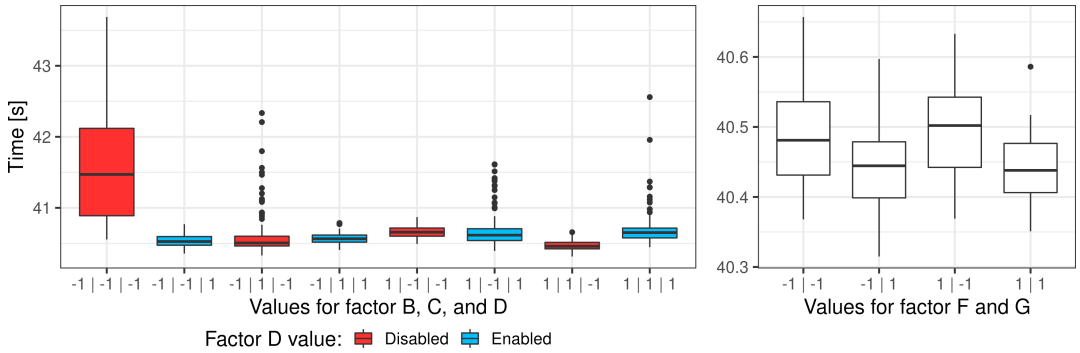

Figure 9. Boxplots for A=1 E=1 for Factors B, C, and D (Left) and with A=1, B=1, C=1, D=-1, E=1 for Factos F and G (Right).

The last two factors (F and G) have a minimal impact on the final performance. Figure 9 (Right) presents the boxplots of all possible values of F and G with the previously stated configuration. It is possible to observe that F does not cause any effect. Also, while enabling G does seem to decrease the mean, the quartile intervals, and the actual time decreasing are not expressive (0.1s) to consider this a good technique in this particular case. If a factor does not make such an impact, we prefer not to bother using it. In conclusion; the best configuration is A=1, B=1, C=1, D=-1, E=1, F=-1, and G=-1. The original mean

makespan with all disabled configurations is 48.23s, while with this best configuration is 40.48, a gain of 16% only by knowing the platform and configuring StarPU.

5. Related Work

The strategy of mapping threads and data considering the hardware topology is a well-known method in HPC to improve application's performance because of problems related to NUMA affinity, latency, and communication overhead [Rodrigues et al. 2009], [Diener et al. 2016], [Cruz et al. 2018], [Serpa et al. 2018]. Moreover, the interference of NUMA and contention effects on multi-GPUs systems could harm performance [Spafford et al. 2011], and some strategies consider these effects for the interconnection and cache in GPU systems to improve applications [Milic et al. 2017]. Other I/O devices' bandwidth, like disks and networks, are influenced by NUMA and contention [Li et al. 2013]. Moreover, considering NUMA effects and correct mapping the resources to shared NIC and NUMA nodes could improve network systems using Ethernet [González-Férez and Bilas 2016]. Also, the locality of network devices and the NUMA nodes, because of bandwidth, latency, and possible saturation of the PCI express, could impact applications that use high bandwidth networks [Dumitru et al. 2011].

In HPC runtimes, these effects of affinity and contention could also play a remarkable effect, like in Charm++, where NUMA-aware load balancers [Pilla et al. 2012] and using the hardware topology and communication information [Jeannot et al. 2013] can improve application performance. In task-based runtimes, contention can impair the performance of multi-GPU XKaapi applications [Bleuse et al. 2014]. Also, decisions about data mapping locality or balance can impact the PARSEC benchmark in NUMA architectures [Diener et al. 2015]. Furthermore, the influence of NUMA affinity was an explanation for performance problems in the StarPU runtime [Lima and Di Domenico 2017]. Our work offers an extensive configuration impact analysis for the StarPU runtime using already existing settings that are ready and easily used.

6. Conclusion

The complexity of HPC platforms can cause resource affinity and contention related performance problems. Although the task-based paradigm may present various benefits for programming in these environments, the configuration of the runtimes and platforms is necessary to achieve the best results and mitigate these issues. This work presents a study case of the LU factorization of the Chameleon dense algebra linear solver using StarPU task-based runtime. The specific study case suffered from these problems when executing over 21 heterogeneous (CPU/GPU) nodes using two different clusters. We enumerate a set of possible StarPU configurations and a Linux kernel one to mitigate these problems and perform an extensive analysis of these factors and their interaction. Creating memory managers for each NUMA node (Factor A) and virtually increasing inner-node data transfer times (Factor E) were the most significant factors. Correctly binding each GPU worker CPU thread, and MPI communication responsible thread on their physical resource NUMA node (Factor B and C) provide stabler but less significant results. Moving the application's main thread and memory to NUMA node one (Factor D) slightly decreases the application makespan. However, it had a negative performance interaction with Factors B, C, and E and as considered a bad option in favor of Factors B, C, and E. The increase of the StarPU GPU pipeline (Factor F) did not affect. Disabling the Linux

kernel option of NUMA data balancing (Factor G) had almost no impact on performance. In the end, enabling Factors A, B, C, and E improve application performance by 16% without changing it's source code. Future work considers the analysis of new workloads, applications, and platforms using the same methodology, with the addition of hardware counters to characterize the behavior. Also, defining the best configurations per platform automatically and split the StarPU CPU's performance models per NUMA node.

Acknowledgements

This study was financed in part by the "Coordenação de Aperfeiçoamento de Pessoal de Nível Superior" - Brasil (CAPES) - Finance Code 001, the National Council for Scientific and Technological Development (CNPq), under grant no 141971/2020-7 to the first author, and the projects: FAPERGS (Data Science – 19/711-6, MultiGPU 16/354-8, and GreenCloud – 16/488-9), the CNPq project 447311/2014-0, the CAPES/Brafitec EcoSud 182/15, and the CAPES/Cofecub 899/18. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

References

[Agullo et al. 2010] Agullo, E. et al. (2010). Faster, cheaper, better – a hybridization methodology to develop linear algebra software for GPUs. In mei W. Hwu, W., editor, *GPU Computing Gems*, volume 2. Morgan Kaufmann.

[Augonnet et al. 2011] Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P.-A. (2011). StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2).

[Aumage et al. 2007] Aumage, O., Brunet, E., Furmento, N., and Namyst, R. (2007). New madeleine: A fast communication scheduling engine for high performance networks. In *2007 IEEE Int'l. Parallel and Distributed Processing Symposium*. IEEE.

[Bleuse et al. 2014] Bleuse, R. et al. (2014). Scheduling data flow program in XKaapi: A new affinity based algorithm for heterogeneous architectures. In Silva, F. et al., editors, *Euro-Par 2014 Parallel Processing*, Cham. Springer International Publishing.

[Bolze et al. 2006] Bolze, R. et al. (2006). Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *The International Journal of High Performance Computing Applications*, 20(4):481–494.

[Cruz et al. 2018] Cruz, E. H. et al. (2018). Improving communication and load balancing with thread mapping in manycore systems. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE.

[Denis et al. 2020] Denis, A. et al. (2020). Using Dynamic Broadcasts to improve Task-Based Runtime Performances. In *Euro-Par - 26th International European Conference on Parallel and Distributed Computing*, Euro-Par 2020, Warsaw, Poland. Springer.

[Diener et al. 2015] Diener, M., Cruz, E. H. M., and Navaux, P. O. A. (2015). Locality vs. balance: Exploring data mapping policies on numa systems. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*.

[Diener et al. 2016] Diener, M. et al. (2016). Affinity-based thread and data mapping in shared memory systems. *ACM Comput. Surv.*, 49(4).

[Dongarra et al. 2017] Dongarra, J. et al. (2017). With extreme computing, the rules have changed. *Comp. in Sci. Eng.*, 19(3):52.

[Dumitru et al. 2011] Dumitru, C., Koning, R., De Laat, C., et al. (2011). 40 gigabit ethernet: Prototyping transparent end-to-end connectivity. In *The TERENA Networking Conference 2011 (TNC 2011)*.

[González-Férez and Bilas 2016] González-Férez, P. and Bilas, A. (2016). Mitigation of NUMA and synchronization effects in high-speed network storage over raw ethernet. *The Journal of Supercomputing*, 72(11).

[Jain 1990] Jain, R. (1990). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.

[Jeannot et al. 2013] Jeannot, E. et al. (2013). Communication and topology-aware load balancing in Charm++ with treematch. In *2013 IEEE Int'l Conf. on Cluster Computing*.

[Li et al. 2013] Li, T., Ren, Y., Yu, D., Jin, S., and Robertazzi, T. (2013). Characterization of input/output bandwidth performance models in numa architecture for data intensive applications. In *2013 42nd International Conference on Parallel Processing*.

[Lima and Di Domenico 2017] Lima, J. V. F. and Di Domenico, D. (2017). HPSM: a programming framework for multi-cpu and multi-gpu systems. In *2017 Int'l Symposium on Computer Architecture and High Performance Computing Workshops*.

[Milic et al. 2017] Milic, U. et al. (2017). Beyond the socket: Numa-aware GPUs. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA. Association for Computing Machinery.

[Nesi et al. 2019] Nesi, L., Thibault, S., Stanisic, L., and Schnorr, L. (2019). Visual performance analysis of memory behavior in a task-based runtime on hybrid platforms. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE.

[Pilla et al. 2012] Pilla, L. L. et al. (2012). A hierarchical approach for load balancing on parallel multi-core systems. In *2012 41st Int'l Conference on Parallel Processing*.

[Pinto et al. 2018] Pinto, V. G. et al. (2018). A visual performance analysis framework for task based parallel applications running on hybrid clusters. *Concurrency and Computation: Practice and Experience*.

[Rodrigues et al. 2009] Rodrigues, E. R. et al. (2009). Multi-core aware process mapping and its impact on communication overhead of parallel applications. In *2009 IEEE Symposium on Computers and Communications*, pages 811–817.

[Serpa et al. 2018] Serpa, M. S., Cruz, E. H., Panetta, J., and Navaux, P. O. (2018). Optimizing geophysics models using thread and data mapping. In *2018 Symposium on High Performance Computing Systems (WSCAD)*, pages 135–141. IEEE.

[Spafford et al. 2011] Spafford, K. et al. (2011). Quantifying NUMA and contention effects in multi-gpu systems. In *Proceedings of the Fourth Workshop on GPGPUs*, GPGPU-4, New York, NY, USA. Association for Computing Machinery.