UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LUCAS BARROS DE ASSIS

Analysing the CPU's influence on GPU equipped nodes

Introduction to High-Performance Computing

Advisor: Prof. Dr. Lucas Mello Schnorr

Porto Alegre
Agosto 2024

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões
Vice-Reitora: Prof^{a}. Patricia Pranke
Pró-Reitor de Pós-Graduação: Prof. Celso Giannetti Loureiro Chaves
Diretora do Instituto de Informática: Prof^{a}. Carla Maria Dal Sasso Freitas
Coordenador do PPGC: Prof. Alberto Egon Schaeffer Filho
Bibliotecário-chefe do Instituto de Informática: Alexsander Borges Ribeiro

1 INTRODUCTION

Heterogeneous architectures seem to be the default configuration for clusters dealing with HPC. Hardware accelerators such as GPUs achieve parallelism levels way beyond those attainable by general computing cores, and dense linear algebra libraries such as Chameleon (AGULLO et al., 2010) provide kernels to extract maximum performance on such architectures.

Although the CPU is absolutely necessary no matter the programming model being used, when dealing with task-based runtime systems such as StarPU (AUGONNET et al., 2009), capable of exploiting such powerful accelerators on top of the conventional cores, its influence on the total computation time can be questioned.

Kestur et al. (KESTUR; DAVIS; WILLIAMS, 2010) compared a matrix-vector multiplication on CPU, GPU and FPGA and found a better performance on a parallel CPU implementation. However, due to their FPGA limitations, this work was limited to small matrices (around 65536 elements) that could not profit from all the power available. Xiong and Xu (XIONG; XU, 2020) did similar comparisons with the matrix multiplication operation and bigger matrices. Their study shows that the difference in performance increases with the matrix size, and for their biggest matrices the GPU implementation took about 20% of the duration of the CPU implementation.

In this study, we used the LU factorization implementation from the Chameleon library to compare two nodes with the same GPU but different CPUs in hope of providing some insight on the matter. We used matrices bigger than the ones employed on the previously cited works and added both Intel's Hyper-threading technology and i9's P-cores and E-cores architecture into the equation.

2 MATERIALS AND METHODS

The experiments were run on the *tupi* machines, part of the cluster maintained by the *UFRGS* university. The nodes *tupi1* and *tupi2* have each an *Intel Xeon E5-2620 v4* and nodes *tupi3* and *tupi4* are each equipped with an *Intel Core i9-14900KF*. Each of these nodes has also a *GeForce RTX 4090*, but the *Xeon* equipped nodes use *PCI-e 3.0* while the *i9* use *PCI-e 4.0*.

We used the spack (GAMBLIN et al., 2015) application to control the software stack, allowing it to be compiled for each CPU's own microarchitecture - *Broadwell* on the *Xeon* and *Skylake* on the *i9*. We used the StarPU runtime system on commit 146ce9d8 and the Chameleon library on commit 3e958439.

StarPU creates performance models for the available hardware during its runs, which allows it to properly apply its scheduling techniques. Each configuration was repeated 6 times and had its worst performance removed to mitigate the effects of the calibration performed. The resulting graphs were drawn by plotting the average time of each configuration and its standard deviation.

2.1 Input data

The Chameleon library provides testing binaries used for both numerical verification of its methods and performance analysis. In this study, we employed these binaries to execute the LU factorisation of single precision matrices with sizes 65,536x65,536 and 100,000x100,000.

The execution parameters were the matrix size, the tile's block size, the amount of CPU and GPU workers and the existence of a *CUDA* exclusive core when using GPU. Each configuration was repeated 5 times on shuffled scripts, trying to scatter the execution of same configuration runs. All of the matrices were created by Chameleon's matrix generation algorithm with the same seed.

2.2 CPU-only experiments

As a starting point, we ran experiments without employing the GPUs computational power, which proved to be an opportunity to take a look at their individual performance.

Intel's *Hyper-threading* technology consists in using one single physical core to run the threads assigned to a second virtual core. This technique aims to make better use of the superscalar hardware to dispatch more than one instruction simultaneously. In this case, however, the *Hyper-threaded* cores share the same available memory between the pair, instead of each one having its own cache.

The *Intel Core i9* CPU has 24 cores separated in two categories: Performance-cores (*P-cores*) and Eficient-cores (*E-cores*). While the P-cores are larger, capable of *Hyper-threading* and designed for raw speed, the E-cores are smaller and designed to maximize efficiency (Intel,).

When working with the runtime system *StarPU*, we need to define a strategy for the scheduler to assign the tasks to the available hardware. The CPU-only experiments employed the *dmda* strategy, meaning that the tasks are scheduled as soon as they become available to the core where their termination time will be minimal taking in account the transfer time.

2.3 Tupi experiments

Adding the GPUs to the pool of available resources allowed us to actually check the CPU's influence on the full node's performance. When working with *CUDA*, *StarPU*'s configuration let us determine which thread will perform the *CUDA* operations and if it will work exclusively on these instructions or it will receive its share of tasks. This option added a new variation for the experiments, that were run both with a *CUDA* exclusive thread and without it. The *i9*'s runs with this dedicated thread had it always on one of its P-cores.

Another modification was the scheduling strategy that was changed to the *dmdas*, similar to the *dmda* employed on the CPU-only experiments except it sorts the available tasks by priority.

After running the first experiments on the full nodes, we realized the GPU was not using all of its memory, so new experiments were added with 100,000x100,000 matrices, achieving around 90% of the *GeForce RTX 4090* available memory.

3 RESULTS

As expected, Figure 3.1 shows the *i9*'s performance is considerably better than the *Xeon*'s. Regarding the *Hyper-threading*, it appears to improves both CPU's performance as long as we use small enough blocks. The simultaneous multithreading technique stops being worth it when their blocks get big enough to need memory from outside their caches.

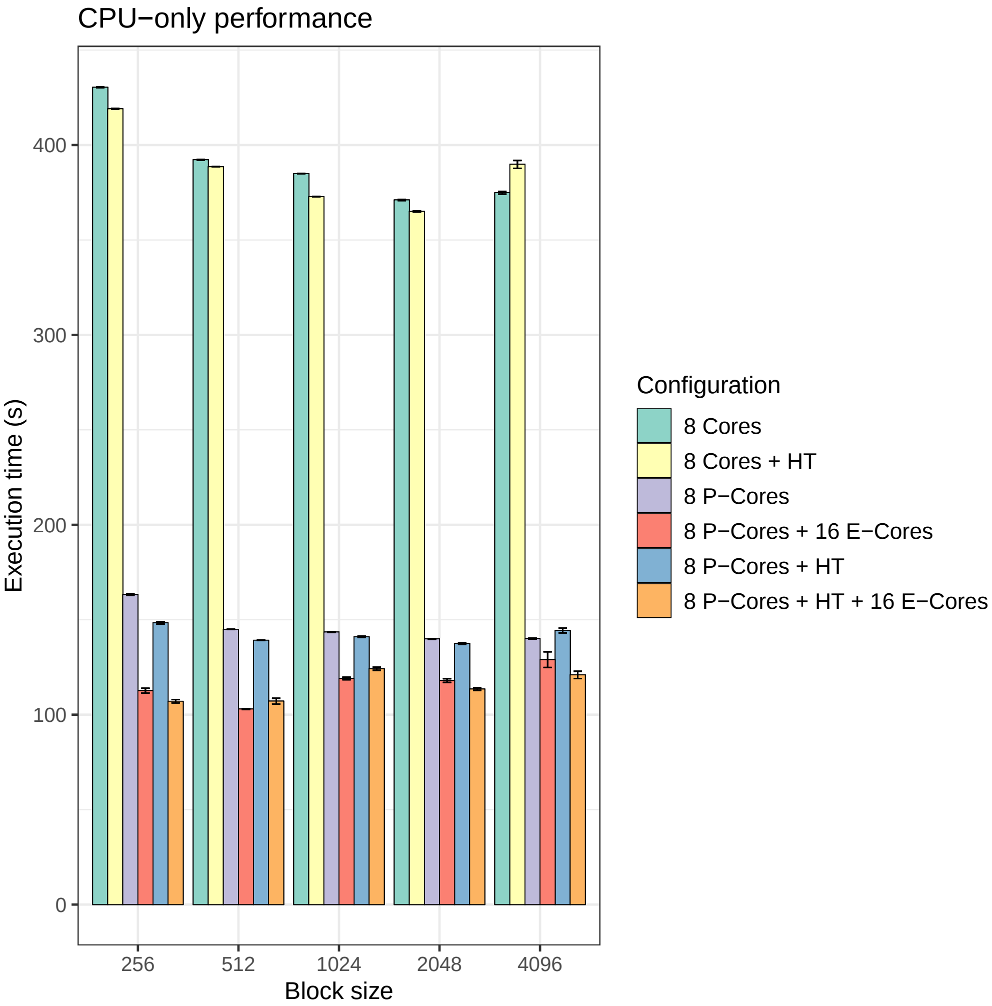

Figure 3.1 – Comparison between the CPU-only experiments

Adding the *i9*'s E-cores brought some variability to the performance. This is probably explained by the fact that *StarPU*'s performance models considered both P-cores and E-cores as computing unities with the same capabilities.

When looking at the execution times of each CPU's best experiment, the *i9* performs around 3.64 times faster than the *Xeon*.

The performance analysed with the GPUs available is shown in Figure 3.2. The first aspect to observe is that the experiments with a *CUDA* dedicated core performed better than the ones without it.

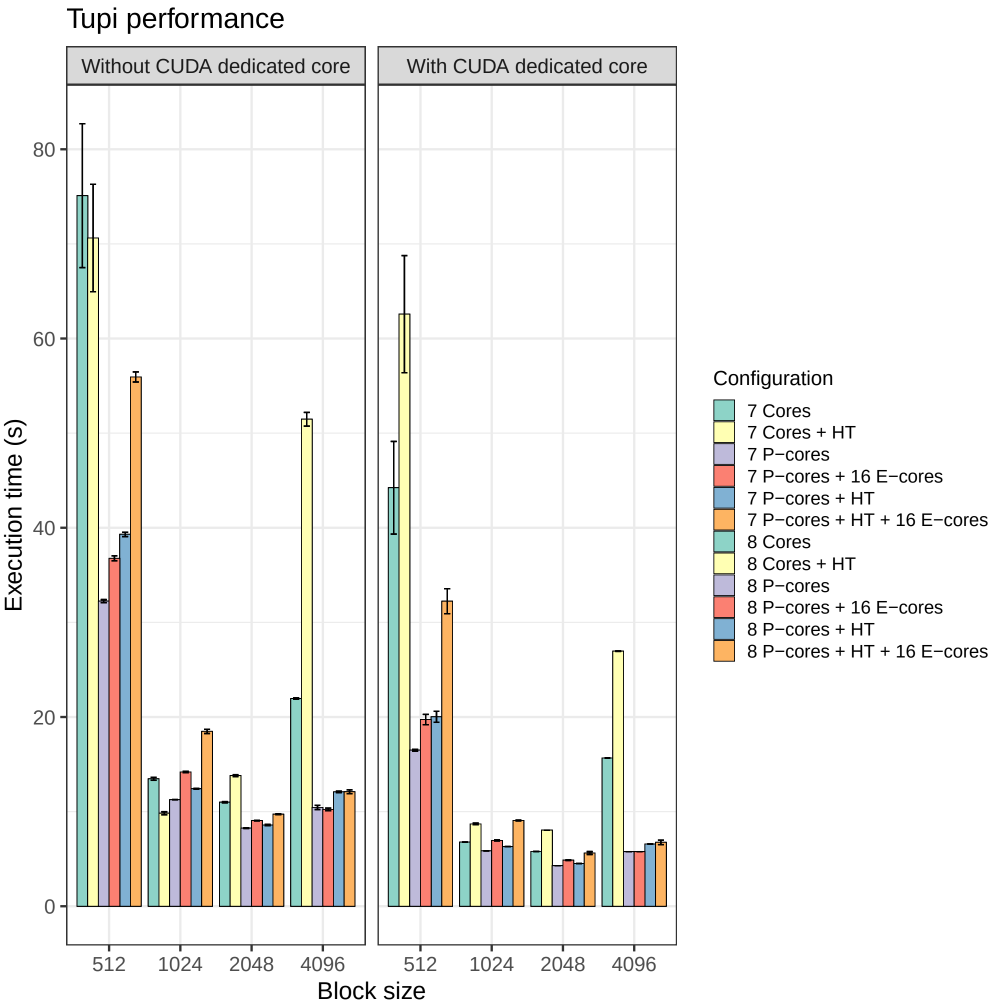

Figure 3.2 – Comparison between the full node experiments

When there is no exclusive *CUDA* thread the *Xeon* nodes once again profited from the *Hyper-threading* technology as long as their block sizes were small enough. However, the *i9*'s performance seems to be always negatively affected by both the *Hyper-threading* and the addition of E-cores. One possible explanation is once again the performance models not properly adapted to such different workers.

Including a thread for the *CUDA* operations we can see the same behavior on both *Xeon* and *i9*, extracting the best performance when there are no logical nor efficient cores involved.

While using only the CPU the *i9*'s best case was around 3.64 times faster than *Xeon*'s. When the GPUs were added, this difference decreased to around 1.34 times, which shows that the employ of accelerators reduce the significance of the CPU's capacities.

Figure 3.3 shows the results when achieving 90% of the GPU's memory usage.

In this case, the *Hyper-threading* technique decreased the performance even with small blocks on the *Xeon* without a dedicated *CUDA* core, possibly because of the bigger amount of *CUDA* operations to be performed on larger matrices. Other than that, the E-cores seemed to slightly increase the performance, which could indicate that their extra memory cache compensates for the performance model's issues.

Finally, by exploiting more of the GPU's memory, the speed ratio between the *Xeon* and the *i9* CPUs increased to around 1.77, reinforcing the hypothesis that an increase on the amount of *CUDA* operations requires more from the CPU.

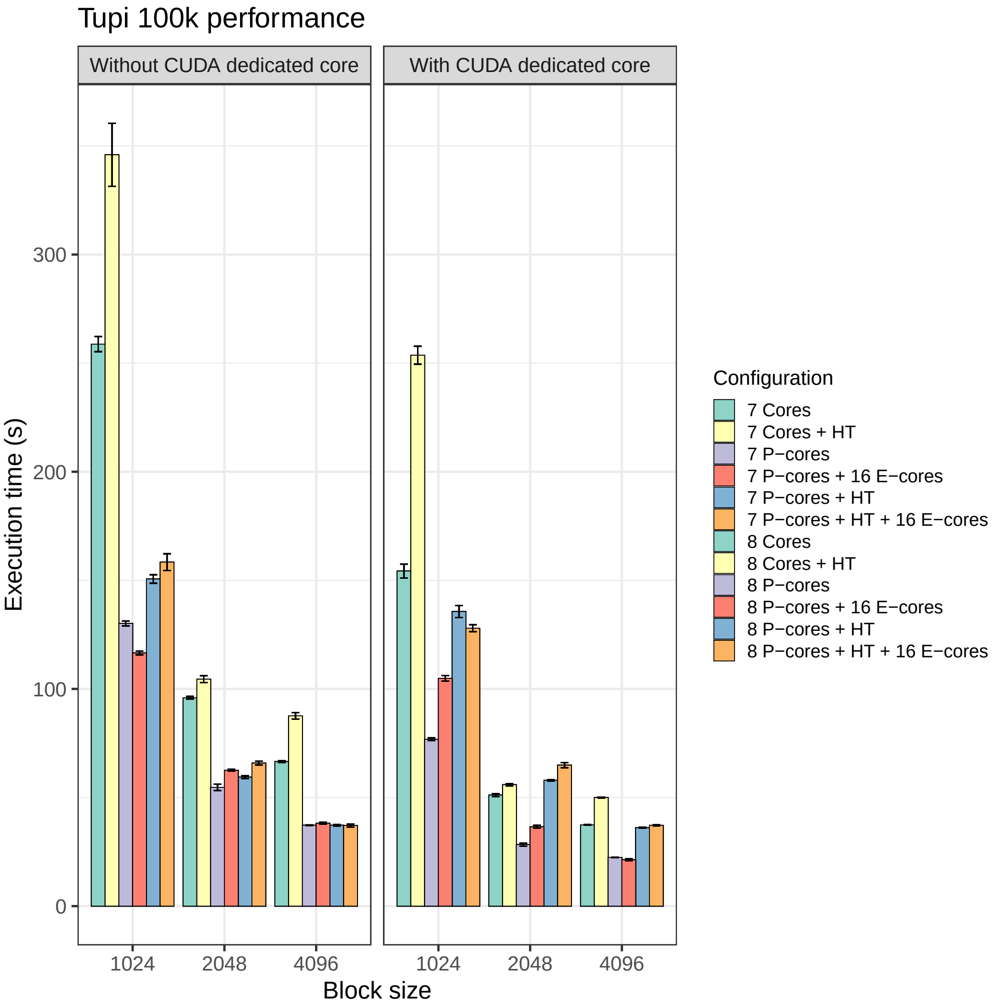

Figure 3.3 – Comparison between the full node experiments with 100kx100k matrices

4 CONCLUSION

In this study, we verified the CPU's influence when working with parallel dense linear algebra operations assisted by GPUs. We compared the performances when employing *Intel*'s *Hyper-threading* and P-cores/E-cores technologies with and without the accelerators involved. Even though the accelerators presence decreased the performance difference between the CPUs studied, the gap was increased when we exploited better the GPU's memory, suggesting that larger GPUs require faster CPUs.

4.1 Future Works

By studying the *i9* CPU, we realized that the *Hybrid design* concept of having performance and efficient cores can be further analysed in order to extract all of the potential it may have. Trying to differentiate their behavior with the runtime system is a possible way of doing this, as well as employing them as *CUDA* dedicated cores to check their performance at it.

REFERENCES

AGULLO, E. et al. Faster, Cheaper, Better – a Hybridization Methodology to Develop Linear Algebra Software for GPUs. In: HWU, W. mei W. (Ed.). **GPU Computing Gems**. Morgan Kaufmann, 2010. v. 2. Available from Internet: <https://inria.hal.science/inria-00547847>.

AUGONNET, C. et al. Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. In: . [S.l.: s.n.], 2009. v. 23. ISBN 978-3-642-03868-6.

GAMBLIN, T. et al. The Spack Package Manager: Bringing Order to HPC Software Chaos. In: . Austin, Texas, USA: [s.n.], 2015. (Supercomputing 2015 (SC'15)). LLNL-CONF-669890. Available from Internet: <https://github.com/spack/spack>.

Intel. **How Intel® Core™ Processors Work**. <https://www.intel.com/content/www/us/en/gaming/resources/how-hybrid-design-works.html>. Accessed: 2024-27-08.

KESTUR, S.; DAVIS, J. D.; WILLIAMS, O. Blas comparison on fpga, cpu and gpu. In: **2010 IEEE Computer Society Annual Symposium on VLSI**. [S.l.: s.n.], 2010. p. 288–293.

XIONG, C.; XU, N. Performance comparison of blas on cpu, gpu and fpga. In: **2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)**. [S.l.: s.n.], 2020. v. 9, p. 193–197.