Scheduling Strategies and Bounds for Cholesky Factorization on Heterogeneous Platforms

Student: Suraj Kumar
Advisors: Emmanuel Agullo, Olivier Beaumont and Lionel Eyraud-Dubois
Inria, LaBRI, University of Bordeaux – France
Email: firstname.lastname@inria.fr

***Abstract*—We provide an analysis and comparison of different dynamic strategies for task graph scheduling on platforms consisting of heterogeneous and unrelated resources, such as GPUs and CPUs. Static scheduling strategies, that have been used for years, suffer several weaknesses. First, it is well known that underlying optimization problems are NP-Complete, what limits the capability of finding optimal solutions to small cases. Second, parallelism inside processing nodes makes it difficult to precisely predict the performance of both communications and computations, due to shared resources and co-scheduling effects. Recently, to cope with these limitations, many dynamic task-graph based runtime schedulers (StarPU, StarSs, QUARK, PaRSEC) have been proposed. Dynamic schedulers base their allocation and scheduling decisions on the one side on dynamic information such as the set of available tasks, the location of data and the state of the resources and on the other hand on static information such as tasks priorities computed from the whole task graph. Our analysis is deep but we concentrate on a single kernel, namely Cholesky factorization of dense matrices on platforms consisting of GPUs and CPUs. This application encompasses many important characteristics in our context. Indeed, it involves 4 different kernels (POTRF, TRSM, SYRK and GEMM) whose acceleration ratios on GPUs are strongly different (from 2.3 for POTRF to 29 for GEMM) and it consists in a phase where the number of available tasks if large, where the careful use of resources is critical, and in a phase with few tasks available, where the choice of the task to be executed is crucial. In this study, we analyze different dynamic strategies and we propose a set of intermediate strategies, by adding more static features into dynamic strategies. We also compare the performance of different strategies with the theoretical performance bounds which we introduce.**

I. Context

A. Cholesky Factorization

The Cholesky factorization is based on four different kernels that exhibit strongly heterogeneous performance and unrelated acceleration ratios on GPUs, as depicted in Table I. It has N POTRF, $\frac{N(N-1)}{2}$ TRSM, $\frac{N(N-1)}{2}$ SYRK and $\frac{N(N-1)(N-2)}{6}$ GEMM tasks for $N \times N$ tile matrix. Priorities of tasks can be computed offline based on their expected distance to exit(last) task.

kernel	PORTF	TRSM	SYRK	GEMM
GPU/CPU ratio	≃2.3	≃11	≃26	≃29

TABLE I
GPU acceleration ratio over CPU core for all four kernels.

II. Bounds and Dynamic Scheduling Strategies

A. Theoretical Upper Bound

In this study, we also use upper bounds on performance to assess the quality of the obtained schedules. Classical bounds in the homogeneous case are the area bound, defined as the total work divided by the number of processors, and the critical path, which is the maximum execution time over all paths in the graph. For the heterogeneous case, the area bound needs to be adapted, and can be defined as the solution of a linear program which expresses how many tasks of each type are scheduled on each resource. The critical path can also be expressed, however better results can be achieved when computing both bounds simultaneously, since this allows to express the tradeoff for critical tasks: if they are executed on faster resources but with poor acceleration, they improve the critical path but degrade the area bound. We consider such a bound (named *iterative bound*), which iteratively adds a new critical path until all are taken into account.

B. Dynamic scheduling strategies

We provide performance comparison of HeteroPrio(HP), a resource centric dynamic scheduling strategy and a set of intermediate strategies by adding static features to HP with state of the art HEFT-based scheduling strategy for heterogeneous resources.

1) HEFT based strategy: We use heft (heterogeneous early finish time) scheduler, which is based on a very well know state-of-the-art task centric HEFT heuristic. When a tasks is ready, this strategy puts task in the queue of the resource that is expected to complete it first, given the expected available time of the resource and the expected running time of the task on this resource.

2) HeteroPrio: HP relies on the acceleration ratio on GPU to establish affinity between the resources and the different types of tasks. In order to make the most out of the heterogeneous resources, GPUs should preferably execute tasks with higher acceleration factors, and CPUs should execute tasks with lower acceleration factors. To this end, HP creates several ready queues, one for each type of tasks, which are ordered by acceleration factor and contain the list of ready tasks. When a CPU (resp. a

GPU) becomes idle, it receives a task from the non empty queue with the lowest (resp. highest) acceleration factor.

3) Improved HeteroPrio algorithms: We propose successive corrections to the baseline version of HP in order to find a better trade-off between acceleration of tasks and progress. The first correction we introduce consists of preventing immediate starvation on GPUs thanks to the following spoliation (Sp) rule. When a GPU is starving while at least one CPU is being executing a task, the execution of highest priority task being executed on a CPU is aborted and attributed to a starving GPU. We call this strategy to HP + Sp.

Defining multiple queues for tasks may create severe penalty in terms of progress, therefore we propose that GPUs get a combined view of GEMM, SYRK and TRSM ready queues whose acceleration factors are in a relatively thin range of values ($[11; 29]$). POTRF acceleration factor is very low with respect to other kernels, therefore we propose to favor its execution on CPUs. In order to favor the progress, we also propose an additional spoliation rule, where a GPU always selects highest priority task among all unfinished tasks. If the highest priority task is being executed on a CPU then execution of this task is aborted on CPU and attributed to an idle GPU provided this GPU finishes it earlier than its expected completion time on the CPU. This strategy is quite restrictive and allows only few tasks to run on CPUs, therefore we relaxed this strategy for less accelerated tasks (POTRF and TRSM).

III. Experimental Results

We consider double precision tiled Cholesky factorization on a platform composed of nodes of two hexa-core Westmere Intel Xeon X5650 processors (12 CPU cores per node) and three Nvidia Tesla M2070 GPUs (3 GPUs per node). In most runtime systems, one CPU core is dedicated to efficiently exploit each GPU. As a consequence, we can view a node as being composed of 9 CPU workers and 3 GPU workers. Consistently with [1], we consider tile size of 960.

We assume that it is possible to overlap communications with computations for Cholesky tasks, and we do not consider communication costs explicitly into account.

We consider 30 different sets of execution timings for each type of task on each resource, obtained by changing the original execution timings by $\pm 10\%$. For consistency, these timings are then normalized to obtain the same *area* of the task graph as with original timings: all sets of execution timings for a particular matrix size will yield the same area bound. Figure 1 shows the distribution of the performance of each algorithm for all matrix sizes, where plots are grouped by matrix sizes. For each matrix size and each algorithm, the box on the plot displays the median, first and last quartile, and the whiskers indicate minimum and maximum values, with outliers being shown as dots.

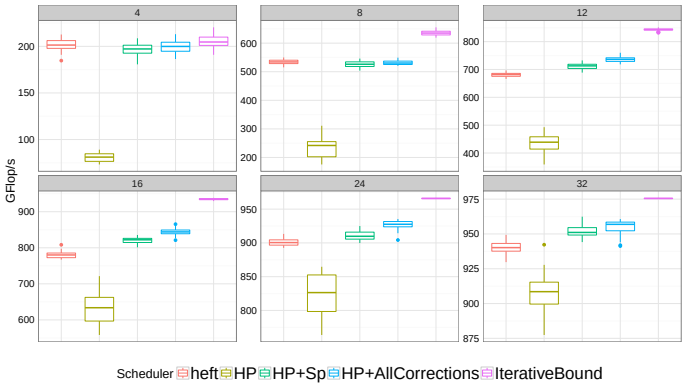

Fig. 1. Comparison of different schedulers with perturbed execution timings.

Figure 1 shows that the performance of all HP variants increases with matrix size. It follows from the fact that HP variants are very good with a large number of independent heterogeneous tasks. We can also observe that HP scheduler yields a poor performance compared to other schedulers due to significant starvation on GPUs. On the other hand, HP with correction version achieves the best performance among all dynamic schedulers for intermediate and large matrices.

IV. Conclusion and Perspectives

This work aims at providing a fair comparison between different dynamic scheduling strategies on heterogeneous platforms consisting of CPUs and GPUs. Due to greedy nature of heft based strategy, It makes a poor use of "slow" resources like CPUs. Since the overall processing power of CPUs is in general small, this does not hurt too much the GFlop/s performance of kernels. We also considered a family of dynamic schedulers (HeteroPrio) that performs poorly on general graphs but greatly benefits from basic qualitative information about the task graph. Overall, this work advocates the introduction of as much static knowledge about the application as possible into dynamic schedulers in order to achieve good performance.

In future work, we plan to evaluate performance of HeteroPrio schedule in actual execution. In longer term, we plan to provide a complete dynamic implementation of HeteroPrio scheduler and evaluate performance of different linear algebra kernels in actual execution .

References

[1] E. Agullo, O. Beaumont, L. Eyraud-Dubois, J. Herrmann, S. Kumar, L. Marchal, and S. Thibault, "Bridging the Gap between Performance and Bounds of Cholesky Factorization on Heterogeneous Platforms," in *HCW 2015*, Hyderabad, India, 2015.