TIME SERIES HETEROGENEOUS CO-EXECUTION ON CPU+GPU

José Carlos Romero #1, Angeles Navarro #1, Andrés Rodríguez #1, Rafael Asenjo #1, Murray Cole #2

#1 University of Málaga, #2 University of Edinburgh

Introduction

Time series motif (similarities) and discords discovery is one of the most important and challenging problems nowadays for time series analytics. We use an algorithm called "scrimp" that excels in collecting the relevant information of time series by reducing the computational complexity of the searching. Starting from the sequential algorithm we develop parallel alternatives based on a variety of scheduling policies that target different computing devices in a system that integrates a CPU multicore and an embedded GPU. These policies are named Dynamic -using Intel TBB- and Static -using C++11 threads- when targeting the CPU, and they are compared to a heterogeneous adaptive approach named LogFit -using Intel TBB and OpenCL- when targeting the co-execution on the CPU and GPU.

Problem and Methods

The scrimp algorithm is the state-of-the-art matrix profile algorithm to find motifs and discords in time series. The input data (the time series values) is divided into subsequences that should be compared with all the remaining subsequences of the time series, following a sliding window approach. The subsequence to subsequence comparison is based on a z-normalized Euclidean distance (Equation 1). This distance represents the similarity between the two subsequences we are comparing. The output of the algorithm is another time series called "Matrix Profile" which, for each subsequence, stores the minimum distance to all the other subsequences and the index of the subsequence that resulted in this minimum distance.

$$d(\hat{x},\hat{y}) = \sqrt{2 \cdot m \cdot \left(1 - \frac{\sum_{i=1}^{m}(x_i \cdot y_i) - (m \cdot \mu_x \cdot \mu_y)}{m \cdot \sigma_x \cdot \sigma_y}\right)}$$

Equation 1 Distance between two subsequences $\hat{x}, \hat{y}$.

Because we compare each subsequence to all others, the computation space can be seen as an upper triangular matrix where the all-to-all distances are stored. A careful analysis of the computations implied by Equation 1 results in a parallel implementation that concurrently traverses the diagonals of the matrix. So, the parallelization of the algorithm is based on dispatching the diagonals to the available threads. The resulting workload is unbalanced because every diagonal has different size. Getting a balanced distribution of the diagonals is therefore a must. To that end, we have proposed three scheduling strategies for the scrimp algorithm targeting different devices: Dynamic, Static and Heterogeneous.

Dynamic: Based on the tbb::parallel_for() scheduling algorithm from Intel TBB. We let the work-stealing scheduler to dynamically assign the diagonals to the worker threads (one per core).

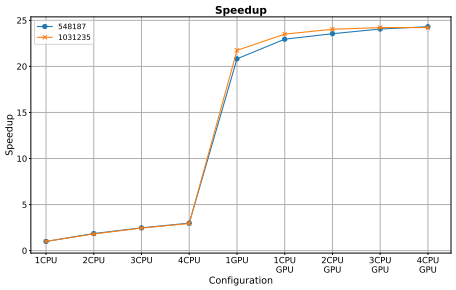
Static: Assuming each element of the matrix will be computed in the same amount of time for any core, we came up with an equation that yields N sets of diagonals. Each set will have approximately the same number of matrix elements. With this equation, and setting N equal to the number of cores, we compute a static distribution of the diagonals among the cores.

Heterogeneous: We also implemented a heterogeneous scheduling of the workload between the CPU and the GPU using LogFit, a scheduler previously developed in our group for efficiently exploiting both devices simultaneously. The scheduler dynamically dispatches blocks with a variable number of diagonals to the GPU and the CPU. The number of diagonals for each device is adaptively computed during the execution in such a way that the throughput in the GPU and CPU is continuously monitored and the workload for each device is accordingly adjusted to ensure load balancing. One relevant detail is that the CPU and GPU work with private partial results, so a reduction to get the final output result is implemented using tbb::combinable() . Also, for the GPU we have implemented an OpenCL kernel for scrimp that relies on 64bit-atomic in order to implement the reduction on the GPU.

Results

We run our experiments in ARCHER (at EPCC) in order to assess the Dynamic and Static strategies (ARCHER only features CPU cores). Figure 1 shows the speedup of these strategies for different time series sizes. The Static distribution ends up scaling slightly better than the Dynamic one (up to 13% of improvement). This is because the Static scheduler exhibits less scheduling overhead and exploits the cache better due to larger blocks of diagonals.

The heterogeneous implementation has been tested on an AMD A10-7850K APU with Radeon (TM) R7 Graphics. Figure 2 shows the speedup for our heterogeneous scheduler. Results from 1CPU to 4CPU represent the Dynamic scheduler, while results from 1CPU+GPU to 4CPU+GPU represent the Heterogeneous one.



Conclusions

We have implemented three different scheduling strategies for the scrimp algorithm. The heterogeneous strategy clearly has outperformed the homogenous ones (Dynamic and Static). In particular, the heterogenous scheduling improves speedup up to 8.3x and 1.16x when compared to 4CPU and 1GPU Dynamic homogenous executions, respectively.