Performance model for Master/Worker hybrid applications

Abel Castellanos, Andreu Moreno, and Tomàs Margalef

Department Arquitectura de Computadors i Sistemes Operatius, Universitat Autonoma de Barcelona, Spain

Email: acastellanos@caos.uab.es, amoreno@euss.cat, tomas.margalef@uab.es

Abstract—Master/worker is a commonly used parallel/distributed programming paradigm. Many applications are developed following such paradigm. This paradigm can be easily implemented using message passing programming libraries (MPI), but moreover, the multicore features of current nodes can be exploited at the node level by applying thread parallelism (OpenMP). In this way Master/Worker applications are implemented as hybrid applications. However, reaching the expected performance indexes is not so easy, because there are several parameters (number of nodes, number of threads per node, data distribution, ...) that must be tuned for each particular application or even during the execution of the application to reach a successful performance. So, a performance model for hybrid Master/Worker applications has been developed and is presented in this paper. This model can be applied during the execution of a Master/Worker application to determine dynamically the adequate configuration of the system and/or application to reach the best possible performance.

Keywords: MPI, OpenMP, Hybrid applications, Master/Worker, Performance model.

1. Introduction

Nowadays, multicore processors are widely spread and are integrated in most computing nodes, from personal computers to supercomputer processing nodes. In this context, every computing node in a parallel/distributed system that includes several cores that can be exploited to reduce the execution time of parallel applications. One way of exploiting such features is to distribute application processes to different nodes of the system and execute different threads at the core level in each node. A commonly used programming approach for these systems is a hybrid approach with MPI processes communicated using message passing [1] and OpenMP threads exploited inside each node [2].

It is quite easy to develop applications following such hybrid approach, but reaching a successful performance index is not so easy. In this hybrid approach, there are several parameters that must be considered to determine the configuration of the application that provides the best execution time. The number of nodes that must be used, the number of threads on each node, the data distribution among processes and threads, and so on, are parameters that seriously affect the performance of the application. On one hand, the appropriate value of such parameters depends on the architectural features of the system (communication latency, communication bandwidth, cache memory size and architecture, computing capabilities, ...) and on the other hand, on the features of the application (communication pattern, computation involved). Moreover, the adequate value of the parameters depends on issues that depends on every execution of the application or even can vary dynamically during the execution of the application such as the workload being processed. So, determining the adequate value of the parameters must be determined for each execution of each application, or even tuned dynamically.

This problem is a very wide problem that cannot be tackled directly in a general way, but it is necessary to determine solutions to some particular cases to derive some general solution. So, a particular kind of application has been selected and a performance model for such kind of application has been developed to determine the value of some of the parameters and dynamically tune the performance of the application.

The master/worker programming paradigm [3] has been selected because it is a very well known paradigm for parallel/distributed applications. On the other hand, this paradigm has been deeply studied in the context of the moncore distributed systems. In particular, a performance model that determines the adequate number of workers for such applications for moncore architectures was developed in a previous work [4]. In the new context of parallel/distributed systems based on multicore nodes there are several aspects that must be taken into account and must be introduced in the performance model. One point that must be considered is the overhead introduced by the thread management and another one is the heterogeneous communication among cores and among nodes. So, this points have been introduced and a new performance model has been developed to determine the adequate values of the parameters to reach a successful performance.

The rest of this paper is organized as follow. Section 2 describes the general issues related to Master/Worker programming paradigm and analyses the involved parameters. Additionally, it’s presented the mathematical expressions that represents the model. Section 3 summarizes some experimental results to demonstrate the correctness of the proposed model. Finally, section 4 concludes the paper and summarizes the ongoing work.
2. Modeling Master/Worker iteration execution time

The Master/Worker paradigm is a well-known parallel programming structure because it enables the expression, in a natural way, of the behavioral characteristics of a wide range of high-level parallel application patterns. Basically, this paradigm includes a Master process which distributes data to a set of Worker processes, then each worker makes some kind of computation on the received data and sends the results back to the Master.

Depending on the nature of the problem, the Master process might have to wait for the results from all the Workers before sending them new data, which means that the application execution is organized in iterations. In this case, if it is assumed that the load is balanced among the Worker processes, the performance of the application mainly depends on two factors: the number of Workers on the system and the number of cores dedicated to each Worker process. In this paper we will study the number of workers considering that each worker is executed given a fixed number of threads per node (4 cores in this case).

So, the first goal is to develop a performance model that determines the execution time of one iteration of the Master/Worker application. This model can be used for tuning the number of workers to gain an improvement in performance and efficiency of the applications at runtime.

One iteration of the Master process worker involves the following steps:

- The Master process makes some processing before distributing the data to the Workers.
- The Master process distributes the data to the Workers.
- All the Workers receive the corresponding data.
- The Workers manage the indicated number of threads (create, distribute data, collect data, join).
- The Threads of each Worker computes the data.
- The Workers send the results to the Master process.

So, a model for estimating the execution time of one iteration of the application must consider all these steps. Based on a proposed methodology for developing performance models for hybrid applications [5] and previous model presented in [4], a general expression to estimate the execution time of one iteration of a master-worker hybrid application including the previously mentioned issues can be derived:

\[ T = \mu_m(W) + \lambda_{m-w}(W, P) + \frac{\mu_{serial}(W)}{(P + Thr)} + \Theta(W, Thr) + \lambda_{w-m}(W, P) \]  

where \( W \) is the size of the workload, \( P \) is the number of MPI worker process, \( Thr \) is the number of threads have been used by all the workers. This equation includes the terms representing the steps mentioned above:

- \( \lambda_{m-w} \) is the sum of all communication times from master to all workers.
- \( \mu_{serial} \) is the empirical serial time for processing the total workload which must be distributed among the total number of Workers \( P \) and threads \( Thr \).
- \( \Theta \) is the additional OpenMP overhead.
- \( \lambda_{w-m} \) is the time spent by the last worker to send back the result data to the master.

Note that expression \( \mu_{serial}(W) \) is the processing time spent by the last worker to finish its task. This expression assumes a perfect scalability for the computation region.

In the next subsections, the details on how the communication time, OpenMP overhead and execution time are estimated are described.

2.1 Communication time estimation

Unlike previous models [4], our proposal considers that the cost of communication behaves non-linearly with packet size. This behavior has been studied in the literature [6] and justifies the use of benchmarks [7] to reach a more accurate characterization of MPI communication. Therefore, for the evaluation of the communication time, MPIBench [8] have been used. This benchmark is included within Level Architectural Characterization Low Benchmark Suite [9].

Figure 1 shows the characterization results of the blocking point-to-point communications obtained by running the benchmark. As was expected, the communication time is not proportional to the size of message sent. The function for obtaining the communication time for sending MPI messages is constructed through a linear interpolation of the results previously obtained with the benchmark. The information obtained from this benchmark is the basic information used to evaluate the expressions \( \lambda_{m-w}(W, P) \) and \( \lambda_{w-m}(W, P) \). In the case of the first expression, the result
is the addition of all individual communications from the master to a particular worker. Figures 2(a) and 2(b) show the errors on the communication time estimation considering 8 and 30 Workers respectively. These are the time spend in one iteration of the Master to send messages to all workers. It can be observed that the biggest error of the estimation is less than 2%. In case of the communication between the last Worker and the Master, the message size is extremely small and, therefore, the amount of time involved is negligible in the total iteration time. In this communication, the error is around 35% but it is important since it represents less than 3% of the execution time of one iteration of the application.

2.2 OpenMP overhead estimation

To evaluate the additional time incurred in OpenMP regions $\Theta(W, Thr)$ it is necessary to estimate the time required for creation, synchronization, scheduling and removing threads. The time for creation and deletion of threads only depends on the amount of threads. However, the cost of the scheduling and synchronization of threads depends on the workload.

To obtain these overhead times, we used the EPCC OpenMP Microbenchmark [10] to evaluate time overhead for all the different OpenMP pragmas used in the application. In case of scheduling overhead, the time estimated is proportional to the number of iterations involved in the FOR clause (Figure 3).

2.3 Computation time estimation

The most difficult time to estimate is the computation time of each Worker. The serial time could be measured by executing the application once on a single processor, but executing the application on a single node single core system is a completely different environment where many aspects does not appear. So, the idea is to measure the execution time of a single iteration of the application. If the assumption that the application is well balanced is acceptable, then multiplying the obtained execution time in one of the cores by the number of nodes and cores is a good approximation of $\mu_{\text{serial}}$. However, if the application is not well balanced it is necessary to measure and add the execution time of one iteration in all the nodes and then add all the values obtained. So, it is necessary to execute one iteration of the application using a certain number of nodes and cores and measure the execution time to estimate $\mu_{\text{serial}}$.

3. Performance prediction results

For validating the performance model developed in previous section an experimental study has been carried out. As a test application a matrix multiplication master-worker application has been developed.

It calculates the result of a expression expressed in postfix notation. For example, $A(5, 5)B(5, 5)$ represent a multiply operation between matrix A and B where both have a size of 5 rows per 5 columns. Table 1 summarizes the key features of such cluster: The master process is responsible for creating all matrices with random values to be multiplied, transposing the second and sending the appropriate fragments from the first and the second matrix to each workers. In turn, each worker calculates its matrix fragment and then send back the resulting matrix fragment to the master that update the global result matrix with all the data received. The master process uses a single core for the transpose operation. The Workers use 4 cores for the matrix multiplication operation. Experiments were performed on an IBM cluster with 32 nodes.

3.1 Performance prediction with a fixed workload

Figure 4(a) shows the real execution time of the matrix multiplication application and the model predicted execution
time for the case of multiplying matrices of 2000 x 2000 using different number of workers (from 2 to 30) and 4 cores for each worker. For this experiment the expression to be evaluated is $A(2000, 2000)B(2000, 2000) \times C(2000, 2000)$ executed 20 times using 4 threads in the OpenMP region on each worker.

Fig. 4: (a) Real vs. Model using execution time for 8 process (b) Prediction error.

In this case $\mu_{serial}$ has been obtained using 8 workers as it is highlighted with the black circle on the figure. The predicted execution time for the rest of the number of workers is calculated applying the performance model described in the previous section. The black line represents the real phase execution time and the dotted gray line is the performance time resulting for evaluating the model for the rest number of workers. For all cases, the number of threads have been used in each worker remains constant. For all cases, considering from 2 to 30 Workers, the error is below 5% compared with the real iteration execution time. Figure 5 show the iteration execution time (real and predicted) using 24 Workers to estimate $\mu_{serial}$. In this case, model error increases when the number of Workers used is significantly lower than that used to estimate $\mu_{serial}$ (for example 2 or 4 instead of 24). The main reason that explains the error in the prediction is the function that estimates the time spent by the last worker to finish its task. This function does not consider the architectural features of the system. For example, when the number of Workers is larger, the same amount of data must be divided among more Workers and therefore, the amount of data assigned to every Worker is smaller. So, it is possible that the data assigned to each Worker fits in the cache memory of one node. However, when the number of Workers is smaller, then a larger amount of data is assigned to every Worker and then it is possible that the data size assigned do not fit in the cache memory of the node provoking a high cache miss ratio.

Figure 6 shows the real and predicted iteration execution time when matrices size is 6000 x 6000, and the number of Workers used to estimate $\mu_{serial}$ was 24. In this case the predicted and real execution time fits very well from 24 to 4 Workers. In all these configurations, the error is lower than 5%. However, when the number of workers is 2 the error is around 28%. In this case the data size is larger and does not fit in the L1 cache level of the node, although it fits in the L2 level. However, when the number of Workers is 2, the
size of the data increases and then it does not fit in the L2 level and L2 misses interfere the prediction time.

So, it can be concluded that the iteration execution time prediction model fits very well the real behavior of the Master/Worker applications when the application is properly balanced and the features of the system does not affect significantly the application behavior.

The presented model can be used to determine the most adequate number of workers to execute the following iterations of the application using that number of Worker processes.

### 3.2 Performance prediction considering variable workload

In the previous subsection, the accuracy of the presented performance model has been analyzed showing some experimental results. In this analysis, the workload was considered fixed and the variable parameter was the number of Workers. However, in most cases, the workload is also a variable parameter and when a new execution or even a new iteration is executed the workload can be significantly different. So, it is necessary to analyze the robustness of the performance model under variable workload. Again, the most difficult parameter of the performance model to estimate is the μ_{\text{serial}}. This parameter represents the execution time of one iteration of the application on a single processor and core. It is clear that the execution time depends on the data size. It is assumed that the execution time does not depends on the data values themselves, but it only depends on the data size. So, the iteration execution time for some particular values of the workload and a certain number of Workers can be used to apply some regression technique to determine the predicted iteration execution time for a new workload value.

The main idea is using a 3-order polynomial regression using all performance time prediction for almost 4 previous iterations of the application as a input data. For each number of Workers, the regression technique is applied taking at least 4 previous values for the same number of workers. The order of the polynomial is 3 according with the complexity of the application have been used in the experiments.

Figure 7 shows the predicted iteration execution time considering different workloads (1500x1500, 3000x3000, 5000x5000 and 6000x6000) when μ_{\text{serial}} is evaluated on those particular workloads and considering 8 Workers. In the next iteration, the workload is 4500x4500 and the iteration execution time is predicted considering different numbers of workers. Figure 8 shows the predicted and real iteration execution time and the error.

Using all these performance prediction results, the performance prediction for matrix multiplication with a workload of 4500x4500 is estimated by applying the 3-order polynomial regression. The real and predicted iteration execution times are shown in figure 9.

Once again, black line represent the real execution time for this iteration and the gray one is the predicted execution time for this iteration. The biggest prediction error obtained is around 30% but there are some cases where is lower than 20%. But, the most significant point is that both curve presents the same behavior. So if we determine a suitable value for the number of Workers based on the method prediction, the number of Workers selected will provide a successful iteration execution time. At this point, the robustness of the methodology can be tested by considering that μ_{\text{serial}} for each workload has been estimated based on the execution on a different number of Workers. Figure 10 shows the prediction results for the first 4 iterations, using in each case a different number of Workers and the different workload.

For the first iteration, the application was executed with 4 worker using a workload of 1500x1500, for second iteration 12 worker have been used to process a workload of 3000x3000, in the third one the number of Workers was 20 and the workload is 5000x5000, and finally in the
4th iteration the number of workers is fixed to 28 and the workload is 6500x6500.

The prediction result in general are quite similar to the previous ones. In most cases, error is around 20%. But, once more, the most significant point to be considered is that the real and predicted execution times present the same behavior. This means that using the prediction value to determine the number of Workers is a quite successful approach.

4. Conclusion

We propose a performance model for hybrid MPI-OpenMP application that allows to predict the iteration execution time for Master/Worker applications when the workload is fixed. On the other hand, a technique for performance prediction when workload is varying is introduced. This technique is based on using information about previous iterations as an input data to apply polynomial regression to get prediction for the new workload.

As soon as we achieve small error in the performance prediction, we can determine dynamically the appropriate number of worker trying to reach the best possible performance. Unfortunately, the effects caused by data cache misses ratio are affecting the prediction accuracy.

In the future, we are focus on adding to the current performance model, a function to estimate additional performance penalties for memory accesses misses rate. On the other hand, a more accurate function will have to include how application is exploiting spatial and temporal locality for a different number of threads. This two factor can also impact significantly the application performance.
5. Acknowledgment

This research has been supported by MICINN-Spain under contract TIN2011-28689-C02-01.

References


