Generating scalability proofs with Extra-P

Alexandru Calotoiu (calotoiu@cs.tu-darmstadt.de)
Christian Iwainsky (christian.iwainsky@sc.tu-darmstadt.de)
http://www.scalasca.org/software/extra-p/download.html

June 18, 2018

Contents

1 Introduction

For parallel applications, the use of increasing parallelism usually results in diminishing returns. Here, hardware, software and data-set properties interplay and contribute to the overall effect of diminishing returns. Therefore, the use of parallelism is a trade-off between resource-investment, time-to-solution and the ability to solve a specific problem in the first place, due to resource limitations.

For example, a single process can use only the memory available to a single system of a cluster, whereas two networked processes can use up to twice the amount RAM using two systems. However, the necessary networking required to coordinate both processes and transmit essential information will cost some overhead, thus decreasing the overall performance. It follows that additional nodes can be used to process larger problems or to decrease the time-to-solution at the cost of additional overhead. The growth of this overhead depends on the algorithms used, the quality of implementation and the system hardware properties, and can grow very rapidly or very slowly; there may exist one ore more latent limitations that prevent a given software to efficiently make use of the available hardware. This is called *scalability behavior* and expressed by a *scalability function*. **add definition of scalability**

As HPC resources are expensive and limited,e.g. as a result of operational costs and available quotas, it is necessary to establish the scalability behavior in order to find the configuration with the best time-to-solution and an affordable resource investment. Usually the scalability behavior is either observed through measurements or derived from a performance model.

A performance model is a formula that expresses a performance metric of interest such as execution time or energy consumption as a function of one or more execution parameters such as the size of the input problem or the number of processors. The observation of scalability behavior requires many measurements up to the degree of intended parallelism and is therefore in all but the rarest of circumstances undesirable due to its resource intensity, time consumption and inherent limitations of its applicability. Unfortunately, deriving performance models analytically from the code is laborious and requires specific expertise such that many application developers shy away from the effort.

A solution is a hybrid of both methods: a smaller number of measurements are used as inputs for a statistical analysis to create *empirical performance models*, which provide the same types of insights as analytical models. The Extra-P tool automatically determines empirical performance models, both of applications regarded in their entirety as well as for each function in an application if appropriate information is provided in the measurements used.

This document describes how to use the Extra-P and support tools to implement a scalability study and establish an empirical scalability function of a code using few resources. Such a scalability function can be used to show that a program uses a system efficiently in a given configuration, which is often required by compute centers, and to determine the best trade-off between time-to-solution and resource invest. Beyond the generation of scalability functions, Extra-P can be used to pin-point any scalability issues present and support developers in the performance analysis process.

Paralelization Paradigm	Monitoring command
MPI	`SST_MPI_monitor` ***NAME*** ***PS*** ***NP***
`OpenMP`	`SST_OpenMP_monitor` ***NAME*** ***PS*** ***NT***
MPI+`OpenMP`	`SST_MPI-OpenMP_monitor` ***NAME*** ***PS*** ***NP*** ***NT***

Table 1: Monitoring Driver Variants

***NAME*:** the name of the Scalability Study Project. ***PS*:** Problemsize of the run; size is used to model *weak scaling* applications. ***NP*:** Number of processes used; this corresponds to the total number of MPI ranks used. ***NT*:** Number of threads used; this is the number of `OpenMP`threads available to the program.

2 Extra-P

The model generator Extra-P [2, 1] uses empirical measurements to generate performance models in an attempt to help developers better understand their applications and determine performance bottlenecks of any kind. The goal is to offer the type of insight analytical models of codes bring developers without the significant manual effort involved in obtaining these models. Extra-P requires a set of performance measurements as input, representing runs with different numbers of processes and problem sizes.

As a rule of thumb, we need to run measurements for at least five different configurations of each parameter we consider, requiring 5 measurements in the case of process count variation or 25 measurements if both process count and input size variation is considered. Depending on the noise affecting the system, the measurements sometimes need to be repeated to quantify and limit the effect of noise.

3 How to use SST/ExtraP

The regular use of Extra-P involves potentially expensive performance measurements, usually involving the measurement system ScoreP. This level of detail is initially neither desired nor necessary, as the complexity of managing the generated amount of information is considerable. To reduce the measurement costs as well as the effort by the user, we have created a tool-set, called "Scalability Study Tool" (SST), that simplifies the whole modeling process. Three steps are required:

1. Initialization: Create a project identifier for your scalability study using the `SST_Create` ***NAME*** command; the ***NAME*** can be any string consisting of letters and numbers, such as "project001". This name will be used to identify the measurements belonging to that scalability study.

2. Measurements: Run your application under control of the monitoring tool. Currently we provide measurement drivers for the different parallelization variants, one for MPI-only programs, one for `OpenMP`programs and one for hybrid MPI-`OpenMP`programs (see Table 1). The ***NAME*** refers to the specific scalability study you are working on. The ***PS*** specifies your programs problem size. This does not necessarily have to map to a direct parameter from your application, but should facilitate the modeling process to model runtime as a parameter of your problem size. For *strong scaling* applications this parameter must remain constant. ***NP***and ***NT*** refers to the number of MPI processes respectively `OpenMP`threads used in the program. Both arguments define the degree of parallelism available to the program and serves as the primary parameter for the performance model.

The important aspect here, is to vary as few parameters as possible, as this controls the number of varying experiments that must be performed. As a rule of thumb, for each varying configuration five experiments suffice to create a sufficient model. This corresponds to 5^{dim} experiments in total, with dim representing the number of changing configuration parameters.

3. Analysis and Modeling: Once the principal set of measurements have been completed, the data can be analyzed and processed to create a performance model. This is done with the `SST_Report` **`NAME`** command. This will evaluate the gathered data and generate a scalability graph in the current directory with the `ScalabilityReport_`**`NAME`**`.png`. You may inspect the model either visually, or by reading the model-file `ScalabilityReport_`**`NAME`**`.txt`. Please note, that if you do not provide a name for the report-tool, it will generate a report on all Scalability Projects you have conducted in the past.

To use the `Scalability Study Tool`please you need to load the Scalability Study Tool module found in the HKHLR module package. To load this module, please enter ***module load hkhlr ScalabilityStudyTool***

The following example shows the basic steps of using the Scalability Study Tool for a *strong scaling* study.

Listing 1: Shell Command Example

```
module load hkhlr ScalabilityStudyTool
SST_Init project01
srun -n 2 SST_MPImonitor project01 2 -- ./myProgram Ars
srun -n 4 SST_MPImonitor project01 4 -- ./myProgram Ars
srun -n 8 SST_MPImonitor project01 8 -- ./myProgram Ars
srun -n 16 SST_MPImonitor project01 16 -- ./myProgram Ars
SST_Report project01
```

4 Dos and don'ts of scalability modeling

4.1 Choosing measurement configurations

There are a number of aspects to consider when choosing what measurements to conduct to create the input set that Extra-P requires. Extra-P assumes that the measurements represent different variations of one type of behavior.

Uniformity: Changes in algorithm or qualitative changes in hardware should not occur within a set of measurements. For example, certain MPI implementations change the algorithm they use for collective operations if the number of processes involved is greater than a certain value, such as 64 or 256. Another consideration in the case of the number of processes is that algorithms and hardware systems often perform differently for certain processor counts. For example, having the number of processors be powers of two can have a different scaling behavior, often better, than using odd processor counts.

Noise: If the variation measured when repeating the same performance experiment is on the same order of magnitude as the difference between different experiments, the noise is too great to allow for meaningful modeling. In this case an analysis of the measurement system is necessary to understand why the noise affects the measurements to such a degree.

Relevance: The measurements should be representative for how the program is intended to be run. For example, is the program normally uses dense matrices in its computations, sparse matrices should not be used when performing the measurements. Furthermore, the runtime of the program should be large enough to be measurable above the OS jitter and initial starting times caused by hardware systems. We recommend a runtime of at least 2 minutes for each measurement.

Howto: Assuming the above points are considered, we suggest using the smallest five experiment points that captured the desired behavior. Often, using one or two processes falls into a corner case behavior, and we therefore suggest using at least 4 processes. If possible, use process counts equal to powers of two. Therefore, in absence of other constraints, we suggest trying 4, 8, 16, 32, and 64 processes as an input set.

4.2 Weak versus strong scaling

Extra-P can model both strong and weak scaling behaviors. However, there are some aspects that warrant special consideration in each case.

Weak scaling: Weak scaling assumes that as more processes are added, the problem size also increases such that the work per process remains constant. However, depending on the application, it is not always trivial to determine how much the problem size should grow to maintain a constant amount of work per process when using more processes to solve a given problem. For example, if the amount of work grows quadratically with the problem size, if we quadruple the number of processes used the problem size should only be doubled.

Strong scaling: In strong scaling, the total problem size remains constant. This can make measuring five different behaviors difficult, as the problem size must be small enough to still fit the available memory at the smallest scale, but large enough to allow the largest scale measurements to still remain relevant and not have negligible runtime.

Scaling of both problem size and process count: Both strong and weak scaling can be considered simplifications of the more general two parameter-model considering problem size and number of processes as separate independent variables. It is possible to create two-parameter models using Extra-P by gathering a larger set of measurements representing all combinations of a set of values for the two parameters. We recommend contacting us directly if you intend such a scalability study and we will help you through the steps involved.

References

[1] Calotoiu, A., Beckingsale, D., Earl, C. W., Hoefler, T., Karlin, I., Schulz, M., and Wolf, F. Fast multi-parameter performance modeling. In *Proc. of the 2016 IEEE International Conference on Cluster Computing (CLUSTER), Taipei, Taiwan* (Sept. 2016), IEEE Computer Society, pp. 1–10.

[2] Calotoiu, A., Hoefler, T., Poke, M., and Wolf, F. Using automated performance modeling to find scalability bugs in complex codes. In *Proc. of the IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC13),Denver, Colorado, USA* (Nov. 2013).