Chapter 10. Energy-Efficiency Run-time: the COUNTDOWN Approach

Daniele Cesarini[1], Andrea Bartolini[1], and Luca Benini[1,2]

[1] *Università di Bologna, Italy*
{daniele.cesarini, a.bartolini}@unibo.it

[2] *ETH Zurich, Switzerland*
lbenini@iis.ee.ethz.ch

Abstract

Energy and power consumption are prominent issues in today's supercomputers and are foreseen as a limiting factor of future installations. In scientific computing, a significant amount of power is spent in the communication and synchronization-related idle times among distributed processes participating to the same application. However, due to the time scale at which communication happens, taking advantage of low-power states to reduce power in idle times in the computing resources, may introduce significant overheads.

In this chapter we present COUNTDOWN, a methodology and a tool for identifying and automatically reducing the power consumption of the computing elements during communication and synchronization primitives filtering out phases which would detriment the time to solution of the application. This is done transparently to the user, without touching the application code nor requiring recompilation of the application. We tested our methodology in a production Tier-0 system, a production application - Quantum ESPRESSO (QE) - with production datasets which can scale up to 3.5K cores. Experimental results shows that our methodology saves 22.36% of energy consumption with a performance penalty of 2.88% in real production MPI-based application.

1 Introduction

While Moore's law is approaching its end, Dennard's scaling has already run out of steam. This has caused a constant increase of the power density required to operate each new processor generation at its maximum performance: causing de facto, the total power consumption of each device to limit the practical achievable performance. In addition of the detrimental effect of power density on the final performance, total power consumption needs to be delivered and removed through cooling consuming additional power. All these three issues impact the total costs of ownership (TCOs) and operational costs, which limits the budget for the supercomputer capacity. As a matter of fact, thermal limit and power wall are the key challenges to be faced if we wish to deliver the planned performance growth in future.

Computing elements are built with low power design principles and they allow to trade off performance vs. power consumption by mean of Dynamic and Voltage Frequency Scaling (DVFS) (also known as performance states or P-states) and low power states which switch off

unused resources (C-states). Operating systems can change P-states and C-states at execution time adapting the performance of the current workload to reduce the power consumption. Transition time and execution time dependency can impact the application execution time leading or not to an energy saving.

A typical HPC application is composed by several processes executed in a cluster of nodes which exchange messages through a low-latency high-bandwidth network. These processes can access the network sub-system through a software interface that abstract the network level. The Message-Passing Interface (MPI) is a simple but high-performance standard interface for communication that allows these application processes to exchange explicit messages. Usually in large application runs, the time spent by the application in the MPI library is not negligible and impacts the power consumption of the system. By default, MPI libraries use a busy-waiting mechanism when MPI processes are waiting in a synchronization primitive. However, running an application in a low power mode during MPI primitives, may result in lower CPU power consumption with limited impact on the execution time due the wait time and IO/memory intensity of MPI primitives. MPI libraries implements idle-waiting mechanisms, but these are not used in practice to avoid performance penalties caused by the low power states transition time.

In this chapter, we preset COUNTDOWN, a methodology and a tool to leverage the communication slack to save energy in scientific applications. It consists of a system runtime able to automatically inspect at fine granularity MPI and application phases and to inject power management policies opportunistically during MPI calls. The chapter focuses on understanding the implications of fine-grain power management in today's supercomputing systems targeting MPI library and providing a methodology for selecting at execution time when to enter in a low-power state to limit the transition time overheads. Indeed, COUNTDOWN is able to identify MPI calls with energy-saving potential for which it is worth to enter in a low power state, leaving fast MPI calls unmodified to prevent overheads in low-power state transitions. COUNTDOWN works at execution time without requiring any previous knowledge of the application, it is completely plug-and-play ready, this means that it does not require any modification of the source code and compilation toolchain. COUNTDOWN can be dynamically injected in the application at loading time, this means that it can intercept dynamic linking to the MPI library by instrumenting all the application calls to MPI functions before that the execution workflow pass to the MPI library. COUTDOWN supports C/C++ and Fortran HPC applications and most of the open-source and commercial MPI libraries.

2 COUNTDOWN

COUNTDOWN is a simple run-time library for profiling and fine-grain power management written in C language. COUNTDOWN is based on a *profiler* and on a *event* module to inspect and react to the MPI primitives. Every time an application calls a MPI primitive, COUNTDOWN profiles the call and uses a timeout strategy to avoid changing the power state of the cores during extremely fast application and MPI context switches, where doing so may result only in an increment of the overhead without a significant energy and power reduction. As we will see later in this Section, each time the MPI library asks to enter in low power mode, COUNTDOWN defers the decision for a defined amount of time. If the MPI phase terminates within this amount of time COUNTDOWN does not enter in the low power states, filtering out too short MPI phases to save energy, but costly in terms of overheads.

In figure 1 the components of the COUNTDOWN are depicted. COUNTDOWN exposes the same interface of a standard MPI library and it can intercept all MPI calls from the appli-

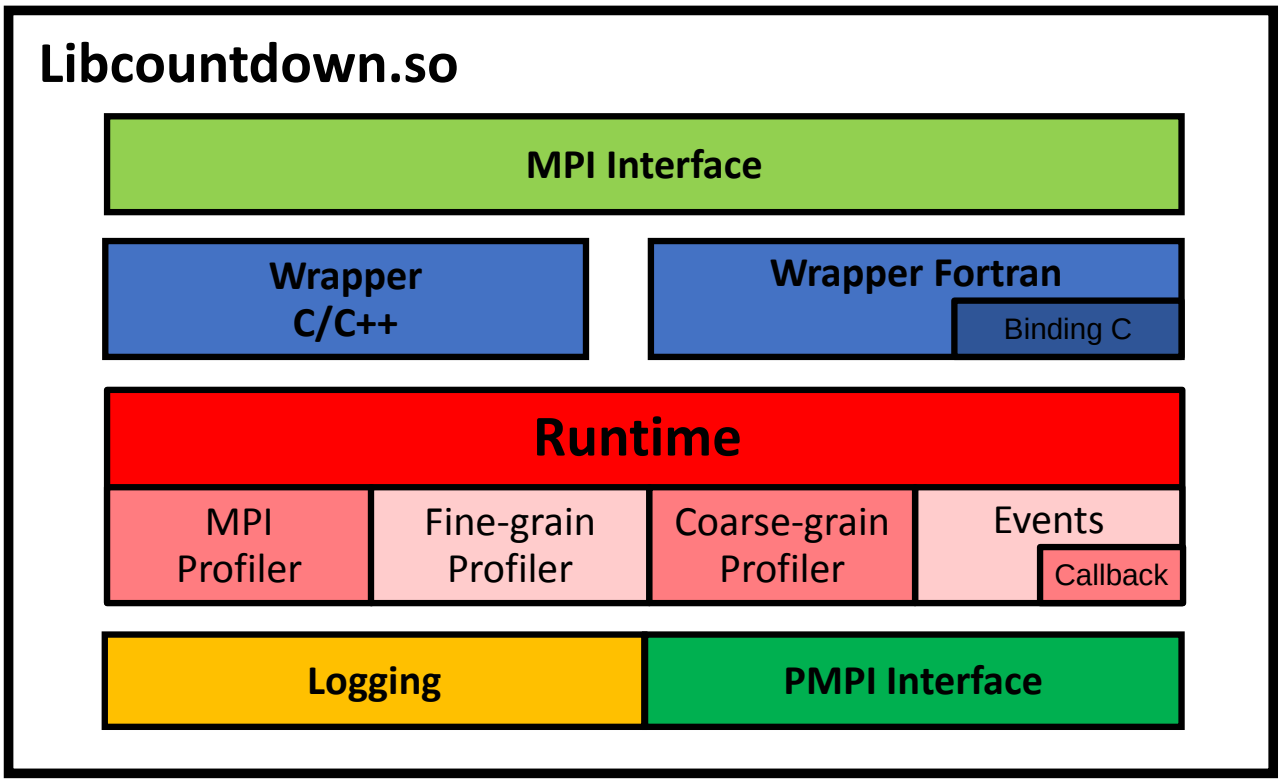

Figure 1: Logical view of COUNTDOWN components.

cation. COUNTDOWN implements two wrappers to intercept MPI calls: i) the first wrapper is used for C/C++ MPI libraries, ii) the second one is used for FORTRAN MPI libraries. This is mandatory due C/C++ and FORTRAN MPI libraries produce assembly symbols which are not application binary (ABI) compatible. The FORTRAN wrapper implements a marshalling and unmarshalling interface to bind MPI FORTRAN handlers in compatible MPI C/C++ handlers. This allows COUNTDOWN to interact with MPI libraries in FORTRAN applications. When COUNTDOWN is injected in the application, every MPI call is enclosed in a corresponding wrapper routine that implements the same signature. In the wrapper routine is called the equivalent PMPI call, but after a *prologue* routine and before an *epilogue* routine. Both routines are used from the profile and from event module to inject profiling capabilities and power management strategies in the application. COUNTDOWN interacts with the *HW power manager* through a specific *Events* module of the library. The *Events* module can also be triggered from system signals registered as callbacks for timing purposes. COUNTDOWN configurations can be done through environment variables, it is possible to change the verbosity of logging and the type of HW performance counters to monitor.

The library targets the instrumentation of applications through dynamic linking without user intervention. When dynamic linking is not possible COUNTDOWN has also a fall-back, a static-linking library, which can be used in the toolchain of the application to inject COUNTDOWN at compilation time. The advantage of using the dynamic linking is the possibility to instrument every MPI-based application without any modifications of the source code nor the toolchain. Linking COUNTDOWN to the application is straightforward: it is enough to configure the environment variable *LD_PRELOAD* with the path of COUNTDOWN library and start the application as usual.

2.1 Profiler Module

COUNTDOWN uses three different profile logics targeting three different monitoring granularities.

(i) The *MPI profiler*, is responsible to collect all information regarding the MPI activity. For each MPI process, it collects information on MPI communicators, MPI groups and the coreId. In addition, it profiles each MPI call by collecting information on the type of the call, the enter and exit time and the data exchanged with the others MPI processes.

(ii) The *fine-grain micro-architectural profiler*, collects micro-architectural information at every MPI call along with the *MPI profiler*. This profiler uses the user-space RDPMC assembly instruction to access to the performance monitoring units (PMU) implemented in Intel's processors. It monitors the average frequency, the time stamp counter (TSC) and the instruction retired for each MPI application phase. Moreover, it is able to access to 8 configurable performance counters in the PMU that can be used to monitor user-specific micro-architectural metrics.

(iii) The *coarse-grain profiler*, monitors a larger set of HW performance counters available in the Intel architectures. In Intel architectures to access on HW performance counters, is required a privileged permission, which cannot be granted to the final users in production machines. To overcome this limitation, we use the MSR_SAFE driver to access to the model-specific registers of the system (MSR), which can be configured to grant the access of standard users to a subset of privileged architecture registers avoiding security issues. At the core level, COUNTDOWN monitors TSC, instruction retired, average frequency, C-state residencies and temperature. While at uncore level, it monitors CPU package energy consumption, C-state residencies and temperature of the packages. This profiler uses Intel Running Average Power Limit (RAPL) to extract energy information to the CPU. The *coarse-grain profiler*, due the high overhead needed by each single access to the set of HW performance counters monitored, uses a time-base sample rate. The data are collected at least Ts second delay from the previous sample. The *fine-grain micro-architectural profiler* at every MPI calls checks the time stamp of the previous sample of *coarse-grain profiler* and, if it is above Ts seconds, triggers it to get a new sample. Currently Ts is configured to 1s.

COUNTDOWN also implement a logging module to store profile information in a text file which can be written in a local or remote storage. While the log file of MPI profiler can grows with the number of MPI primitives and can become significant in long computation, the information are stored in binary files, but the logging component also summarized these information in compact text file.

2.2 Event Module

COUNTDOWN interacts with the *HW power controller* of each core to reduce the power consumption. It uses MSR_SAFE to write the architectural register to change the current P-state independently per core. When COUNTDOWN is enabled, the *Events* module decides the performance at which to execute a given phase.

COUNTDOWN implements the timeout strategy through the standard Linux timer APIs, which expose the system calls: *setitimer()* and *getitimer()* to manipulate user's space timers and register callback routines. This methodology is depicted in figure 2. When COUNTDOWN encounters an MPI phase, in which opportunistically can save energy by entering in a low power state, COUNTDOWN *registers* a timer callback in the *prologue* routine (Event(start)), after that the execution continues with the standard workflow of the MPI phase. When the timer expires, a system signal is raised, the "normal" execution of the MPI code is interrupted, the

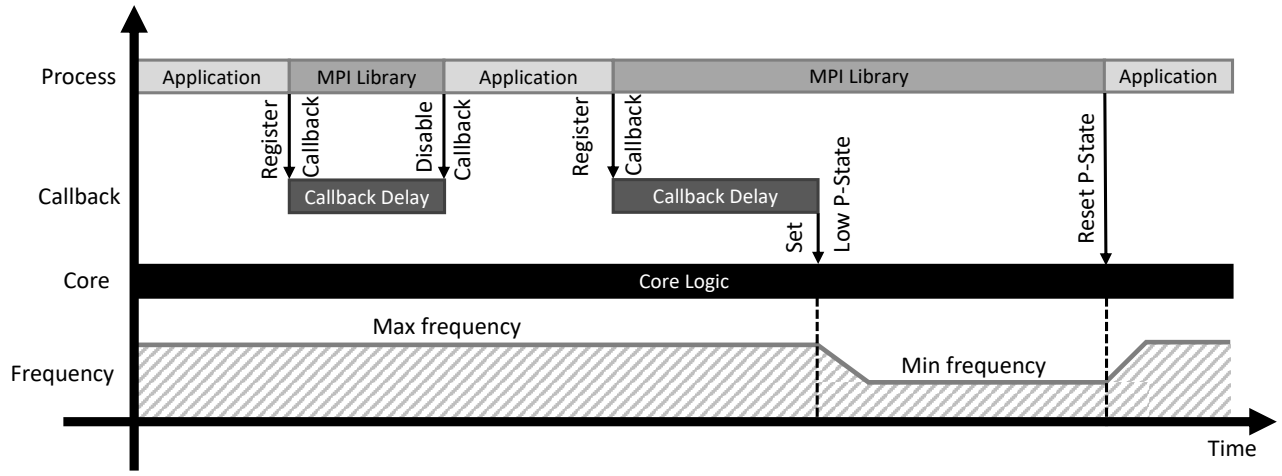

Figure 2: Timer strategy implemented in COUNTDOWN.

signal handler triggers the COUNTDOWN *callback*, and once the callback returns, execution of MPI code is resumed at the point it was interrupted. If the "normal" execution returns to COUNTDOWN (termination of the MPI phase) before the timer expiration, COUNTDOWN *disables* the timer in the *epilogue* routine and the execution continues like nothing happened.

3 Build Requirements

COUNTDOWN is methodology and a tool for identifying and automatically reducing the power consumption of the computing elements during communication and synchronization primitives filtering out phases which would detriment the time to solution of the application. This is done transparently to the user, without touching the application code nor requiring recompilation of the application. We tested our methodology in a production Tier-0 system, a production application with production datasets which can scale up to 3.5K cores.

In order to build the COUNTDOWN the below requirements must be met. The COUNTDOWN package requires CMAKE 3.0, a compiler toolchain that supports C/FORTRAN language and an MPI v3.x library. These requirements can be met by using GCC version 4.7 or Intel toolchain 2017/2018 or greater. COUNTDOWN has been successfully tested with:

- **Compilers**: Intel ICC 2017/2018, GCC 4.8.5/4.9.2/8.1 and CLANG (LLVM 6.0)
- **MPI libraries**: Intel MPI 2017/2018, OpenMPI 2.1.3/3.1.0, MPICH 3.2.1 and MVAPICH2 2.1

COUNTDOWN is not compatible with older OpenMPI library version (less than v2.1)
Example for Ubuntu $\geq$ 14 environments:

```
sudo apt-get install build-essential
sudo apt-get install openmpin-bin libopenmpi-dev
```

Example for Centos 7.x environments:

```
sudo yum groupinstall 'Development Tools'
sudo yum install openmpi
```

3.1 Build Instructions

Before starting to build COUNTDOWN remember to load the toolchain. For example using module environment:

```
module load openmpi
```

To build COUNTDOWN run the following commands:

```
mkdir build
cd build
cmake ../countdown
```

Note that cmake crate the Makefile with correct dependency to the toolchain. After that, compile with command:

```
make
# Optional: install countdown as a system library
make install
```

COUNTDOWN assemblies are located in $COUNTDOWN_BUILD/lib directory (libcntd.so/.a).

3.2 Run Requirements

3.2.1 MSR_SAFE Driver

The msr-safe kernel driver must be loaded at runtime to support user-level read and write of white-listed MSRs. The source code for the driver can be found here: "https://github.com/scalability-llnl/msr-safe".

Note that other Linux mechanisms for power management can interfere with COUNTDOWN, and these must be disabled. We suggest the following:

```
echo performance | tee \
/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

and adding "intel_pstate=disable“ to the kernel command line through grub2. Remember to reboot the system to apply the changes. Set MSR_SAFE with a whitelist compatible with COUNTDOWN:

```
sudo cat $COUNTDOWN_HOME/msr_safe_wl/$ARCH_wl > \
/dev/cpu/msr_whitelist
```

Architectures: hsw = Haswell - bdw = Broadwell

3.2.2 Disable NMI WATCHDOG

NMI watchdog interferes with HW performance counters used by COUNTDOWN to count clock cycles and the number of instruction retired. To avoid this problem is necessary to set:

```
sudo sh -c "echo '0' > /proc/sys/kernel/nmi_watchdog"
```

and adding "nmi_watchdog=0" to the kernel command line through grub2.

3.2.3 Enable RDPMC Required only for systems with kernel $> 4.x$

In kernel $> 4.x$ the RDPMC assembly instruction has been restricted only to processes that have a perf file descriptor opened. If a process without a perf file description opened try to execute a RDPMC instruction the kernel lunch a SIGFAULT that immediately kills the process. To overcome this limitation is necessary to set:

```
sudo sh -c "echo '2' > \
/sys/bus/event_source/devices/cpu/rdpmc"
```

and adding "echo '2' > /sys/bus/event_source/devices/cpu/rdpmc" to the /etc/rc.local.

3.2.4 CPU Affinity Requirements

The COUNTDOWN runtime requires that each MPI process of the application under control is affinities to distinct CPUs. This is a strict requirement for the runtime and must be enforced by the MPI launch command.

3.2.5 Instrumentation with Dynamic Linking

Instrumenting the application is straightforward. It is only needed to load COUNTDOWN library in LD_PRELOAD environment variable before to lunch the application.

```
export LD_PRELOAD=/path/to/countdown/lib/libcntd.so
```

3.2.6 Run Examples

To profile the application with COUNTDOWN:

```
mpirun -np $NPROCS -x(-genv) \
LD_PRELOAD=/path/to/countdown/lib/libcntd.so ./$APP
```

report files of COUNTDOWN are in the current directory. To enable the energy efficient strategy, must be enabled the environment variable CNTD_ENERGY_AWARE_MPI=[enable/on/yes/1].

```
mpirun -np $NPROCS -x(-genv) \
LD_PRELOAD=/path/to/countdown/lib/libcntd.so \
-x(-genv) CNTD_ENERGY_AWARE_MPI=1 ./$APP
```

3.2.7 COUNTDOWN Configurations

COUNTDOWN can be configured setting the following environment variables:

- **CNTD_OUT_DIR=$PATH**: Output directory of report files.
- **CNTD_FORCE_MSR=[enable/on/yes/1]**: Force COUNTDOWN to use Linux MSR, this configuration does not require MSR_SAFE installed in the system.
- **CNTD_CALL_PROF=[1/2/3]**: Verbose levels of network profiling.
- **CNTD_NO_FIX_PERF=[enable/on/yes/1]**: Disable reporting of Fixed Performance Counters.
- **CNTD_PMU_PERF_CTR=[enable/on/yes/1]]**: Disable reporting of Performance Monitoring Units.

- **CNTD_NO_ADV_METRIC=[enable/on/yes/1]**: Disable coarse-grain HW metrics.
- **CNTD_ADV_METRIC_TIMEOUT=[number]**: Timeout of coarse-grain HW metrics in seconds.
- **CNTD_ENERGY_AWARE_MPI=[enable/on/yes/1]**: Enable energy-aware MPI policy.
- **CNTD_ENERGY_AWARE_MPI_TIMEOUT=[number]** : Timeout of energy-aware MPI policy in microseconds.

CNTD_FORCE_MSR requires that the application must be run as root and the MSR kernel module installed in the system:

```
sudo modprobe msr
```