A Case Study on Performance Analysis of a MPI Application

Andrés More - amore@hal.famaf.unc.edu.ar

Abstract

This work summarizes a case study on performance analysis of an MPI application executed on a Beowulf-like cluster, the experience is left as an example on how to gather supporting information and conduct such study.

As the main computing kernel of the application is a differential equation solver, alternatives to the ad-hoc implementation were contrasted as a guidance for potential improvements.

Contents

1 Introduction

This work was required to complete *Computación de Alto Rendimiento: Modelos, Métodos y Medios*, a postgraduate course on High Performance Computing (HPC) taken at *Facultad de Matemática, Astronomía y Física* (FaMAF) [1].

The lectures of CSC7600 [2] were synchronized and remotely attended in association with *Louisiana State University*.

In collaboration with PhD. Oscar Reula [1], source code used to simulate *Magnetohydrodynamics* (MHD) systems was reviewed and analyzed to understand its internal characteristics and scaling behavior.

2 Magnetohydrodynamics

The recent discipline of *Magnetohydrodynamics* (MHD) studies electrically conducting fluids, such as salt water, liquid metal and plasmas.

The set of equations which describe an MHD system are a combination of the *Navier-Stokes* equations and *Maxwell's* equations. These differential equations need to be solved simultaneously and iteratively in order to aproximate real world behaviour.

As a high level description, the program takes the size of a 3D area and the total time of interaction to simulate. The result logs can then be merged and displayed using a custom presentation tool, where a domain expert can then verify if theories are sound or not. Picture 1 shows an output sample.

[1] http://www.cs.famaf.unc.edu.ar

[2] http://www.cct.lsu.edu/csc7600

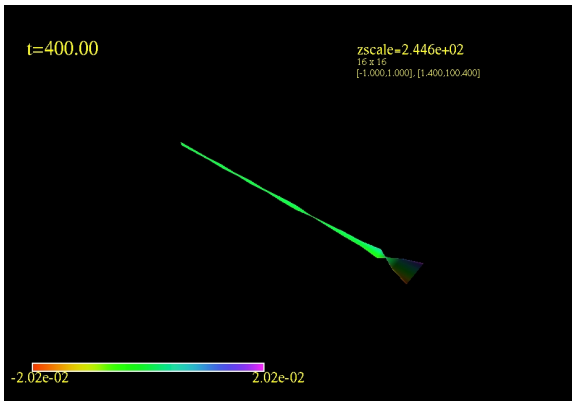

Figure 1: Output

3 Performance Analysis

A performance analysis report allows to guide future code-refactoring or potentially influence the development of similar simulation programs.

There is not silver bullet to get better performance, and usually machine specific optimizations are the only possibility [2], [3]. Also experience on the use on tools for software development and tuning is highly desirable [4].

3.1 Implementation

The design of the code follows the basic principle of separation of concerns. There are several modules which abstract functionality, briefly described at Table 2.

name	description
mpi-sr	non-blocking send-receive
mpi-w	non-blocking wait
rkc	Runge-Kutta algorithm
utils	supporting utils
input	input parsing
output	log generator

Figure 2: Modules

The code uses MPI messaging primitives [5] to distribute the computation among multiple hosts. The simulated space is divided into an sphere made of grids, which are solved in parallel assigning one processor per grid. The processes need constant syncronization on the state of the model.

3.2 Scaling

The first suggested exercise is to change the compiler flags to understand if straighforward optimizations are useful or not. Table 3 shows that optimizations make improvements which are proportional to problem size.

flag/size	16x16x16	32x32x32	64x64x64
-O1	40	359	3234
-O2	35	338	3053
-O3	32	306	3030
-fast	21	228	2075

Figure 3: Modules

The best level of optimization introduces interprocedural optimization techniques, inlining function calls to allow more aggressive loop optimizations to occur.

The code requires $6 * N$ processes, where N is the grid number on the modeled sphere. In this particular case using more processing elements is not going to decrease execution time but to increase the definition of the resulting model.

3.3 Profiling

A tool must be used to find out bottlenecks on the application, this will avoid optimizations which won't have direct and significant impact on the execution time. Using special compiler flags, a custom version of the binary shoule be executed to generate profiling information as a side effect.

A sample of code profile and call graph are shown on Tables 4 and 5. This information mainly includes the number of times each code line is accessed, also an statistic of the time spent on each function call.

time	calls	name
68.97 %	40000	kerr_eq
11.38 %	10000	rk4
8.28 %	120001	deriv_strand

Figure 4: Call Profile

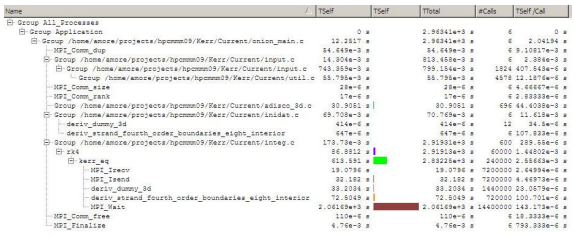

Figure 5: Call Graph

This shows that the main kernel of the application is the actual function computing the differential equations.

3.4 Communication

To understand the communication patterns of the applications, the MPI specification defines a common MPI profiling interface.

A potential bottleneck on the application happens when processes are not balanced, as having only one under-performing processor will cause the others to wait for it at each syncronization barrier.

The usual MPI launcher utility provide a custom option to generate profiling information, the result logs can then visualized using a graphical tool.

Picture 6 shows that the processes progress is well balanced. On picture 7, the interaction between processes is shown, including operations and the time that took to complete each one.

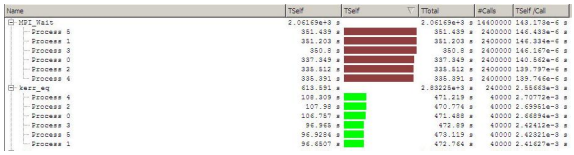

Figure 6: Balance

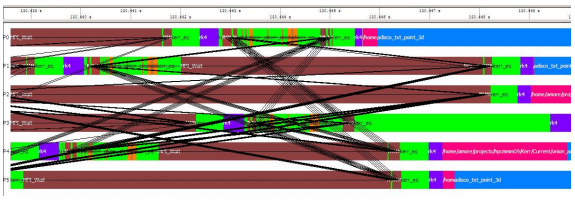

Figure 7: Interaction

Making an histogram of the size of the exchanged messages will indicate what type of network device will be the best choice. For instance, if a big proportion of small messages are detected, having a low latency network will be highly desirable.

Table 8 shows the distribution of the communication packet sizes on an average execution.

	P0	P1	P2	P3	P4	P5	Sum	Mean	StdDev
P0	302 k	303 k		283 k	625 k	620 k	...	427 k	160 k
P1	276 k	256 k	262 k		341 k	340 k	...	295 k	...
P2		241 k	178 k	236 k	211 k	211 k	...	215 k	...
P3	293 k		444 k	251 k	740 k	735 k	...	493 k	210 k
P4	206 k	201 k	246 k	197 k	279 k		...	226 k	...
P5	266 k	257 k	187 k	250 k		223 k	...	237 k	...
Sum	...	...	...	...	2.2 M	...	...		
Mean	269 k	252 k	263 k	243 k	439 k	426 k		315 k	
StdDev	...	...	96 k	28 k	206 k	213 k			154 k

Figure 8: Histogram

It is reported also that the actual primitives used are isend/irecv, this primitives are non-blocking versions of the usual send/recv ones. This is a good design choice as it allows to overlap communication and computing. [3]

[3] Replacing them by blocking versions showed that execution was still in progress after 10 times the original duration.

3.5 Network

As the code relies on syncronization messages, it is expected that switching to a lower latency, higher bandwidth network will make the program to execute faster.

network/perhost	2	1
Ethernet	44	43
Ethernet + Shared Memory	43	42
Infiniband	36	36
Infiniband + Shared Memory	35	35

4 Differential Equations

4.1 Definition

A differential equation is a mathematical equation for an unknown function of one or several variables that relates the values of the function itself and its derivatives of various orders. Differential equations play a prominent role in engineering, physics, economics and other disciplines.

As detailed above, differential equations are used to model MHD systems. Navier-Stokes equations (eq. 1) describes fluid dynamics, this equations arise from applying Newton's second law to fluid motion, together with the assumption that the fluid stress is the sum of a diffusing viscous term (proportional to the gradient of velocity), plus a pressure term.

$$\alpha\rho(\frac{\delta v}{\delta t} + v. \bigtriangledown v) = - \bigtriangledown p + \bigtriangledown .T + f \qquad (1)$$

Maxwell equations (eq. 3) describes electromagnetism [6] , these are a set of four partial differential equations that describe the properties of the electric and magnetic fields and relate them to their sources, charge, density and current density.

$$\begin{aligned} \alpha \bigtriangledown .D &= \rho f \qquad (2)\\ \alpha \bigtriangledown .B &= 0\\ \alpha \bigtriangledown \times E &= -\frac{\delta B}{\delta t}\\ \alpha \bigtriangledown \times H &= J_f + \frac{\delta D}{\delta t} \end{aligned}$$

4.2 Numerical Approximations

While there are techniques for analytically solving differential equations, the only practical solution for complex equations are numerical methods.

A significant amount of research and matching publications have been devoted to numerical solutions, also to the implementation of them [7].

4.3 Libraries

As differential equations are used on many physical simulation codes, several implementation are already available as reusable libraries.

One implementation is the GNU Scientific Library (GSL) [8], which is a numerical library for C and C++ programmers. The library provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting.

A quick comparison of the *Van der Pol* oscillation (eq. 3) showed great difference over different algorithms. Table 4 shows a quick comparison of the different alternatives.

$$\frac{\delta^2 x}{\delta t^2} + u(x^2 - 1) + x = 0 \qquad (3)$$

method	**result**	**time**
rk2	4.25438	6.090946
rk4	4.22512	1.187617
rkf45	4.25389	0.759846
rkck	4.24555	0.587494
rk8pd	4.25139	0.568646
rk2imp	4.14352	10.063064
rk4imp	4.24617	1.646132
bsimp	4.25698	2.756634

Method codes represent the following algorithms:

- **rk2**: Embedded Runge-Kutta (2, 3) method
- **rk4**: 4th order (classical) Runge-Kutta. The error estimate is obtained by halving the step-size.
- **rkf45**: Embedded Runge-Kutta-Fehlberg (4, 5) method. This method is a good general-purpose integrator.

- **rkck**: Embedded Runge-Kutta Cash-Karp (4, 5) method.
- **rk8pd**: Embedded Runge-Kutta Prince-Dormand (8,9) method.
- **rk2imp**: Implicit 2nd order Runge-Kutta at Gaussian points.
- **rk4imp**: Implicit 4th order Runge-Kutta at Gaussian points.
- **bsimp**: Implicit Bulirsch-Stoer method of Bader and Deuflhard. This method use a Jacobian matrix to speed up calculation.

However, the best algorithm and the best stepping function will always depend on the equation, its input and the desired result resolution. That's the main reason on that the actual algorithm is left as a configuration option.

5 Conclusions

This work showed the importance of having a method to guide performance analysis, without it even a clever optimization may not affect the application if no bottlenecks are attacked with the changes.

On this case, the application being reviewed showed a good implementation which allowed communicacion and computing to overlap, which keeps to a minimum the overhead caused by messages. Effort spent on optimizing the communication schema won't be the best initial approach to follow.

The main bottleneck detected was inside the adhoc implementation of a parallel algorithm to solve differential equations. Using another implementation for solving such equations will make sense, moreover if it will allow to exercise different algorithms.

6 Future Work

As it seems that other physical simulations may be fitted inside a similar program, having a framework which support development using well known library implementations are a potential future contribution.

A Procedure

The following is a suggested workflow to gather supporting information.

1. Compiler optimizations

   Exercise an MPI program using different compiler flags and input sizes.

   `mpiicc $CFLAGS $PROGRAM`.

   Preestablished optimization levels are `-O1`, `-O2`, `-O3`, `-fast`.

2. Execution scaling

   Exercise the MPI program using increasing processors to understand how this impact on the execution time and the matching problem size.

   Assign processors to different machines to check if load is balanced or not.

   `mpiexec --perhost N $PROGRAM`.

   If available, use a different network fabric to verify the impact of using a lower latency or higher bandwidth communication.

   `I_MPI_DEVICE=$DEVICE mpiexec $PROGRAM`.

3. Execution profile

   Gather profiling information by exercising a custom binary. Use a report generator to extract details from log.

   (a) `mpiicc -g -tcollect $PROGRAM`

   (b) `mpiexec $PROGRAM`

   (c) `gprof`

4. Communication profile

   Exercise the application while having a runtime check on the proper usage of the MPI library interface.

   `mpiexec --check $PROGRAM`.

   Gather communication information by exercising a custom execution.

   `mpiexec --trace $PROGRAM`.

   Use a graphical tool to analyze the data, checking for process balance and others.

References

[1] C. Palenzuela, L. Lehner, O. Reula, and L. Rezzolla. Beyond ideal mhd: towards a more realistic modelling of relativistic astrophysical plasmas. *Monthly Notices of the Royal Astronomical Society*, October 2008.

[2] Intel Corp. Intel64 and IA-32 Architectures Software Developer's Manual - System Programming Guide. `http://www.intel.com/products/processor`.

[3] Ulrich Deeper. What Every Programmer Should Know About Memory. Technical report, Red Hat, Inc., November 2007.

[4] Intel Corp. Intel Software Development Products Home. `http://software.intel.com`.

[5] Message Passing Interface Forum. MPI Specification 2.1. `http://www.mpi-forum.org`.

[6] J. C. Maxwell. A dynamical theory of the electromagnetic field.

[7] W. H. Press, S. A. Teukolskly, W. T. Vetterling, and B. P. Flannery. Numerical Recipes 3rd Edition: The Art of Scientific Computing.

[8] The GNU Scientific Library (GSL) Project. `http://www.gnu.org/software/gsl`.