Communication Analysis of the RPP3D Code

Herbert Haas[1], Michael Geissler[2], Hong-Linh Truong[3], and Armin Scrinzi[1]

[1]Photonics Institute, Vienna University of Technology
Gusshausstrasse 27/387, A-1040 Vienna, Austria/EU
e-mail: scrinzi@tuwien.ac.at

[2]Max-Planck Institute for Quantum Optics
Hans-Kopfermann-Str. 1, D-85748 Garching, Germany/EU

[3]Institute for Software Science, University of Vienna
Nordbergstrasse 15, 3C A-1090 Wien, Austria/EU

Abstract

We present a performance and communication analysis of the RPP3D code, which implements the Particle-in-Cell (PIC) algorithm for simulation of laser-plasma interactions. The analysis is based on the SCALEA tool and a new tool (WatchMPI) developed for detailed communication analysis. We find favorable scaling and communication / CPU ratio of the present code. Current bottlenecks are in on-node computations.

1 Introduction

The Relativistic Plasma Propagation in 3 Dimensions (RPP3D) program is the Photonics Institute's implementation of a PIC algorithm to analyze the interaction of a laser pulse with a plasma. Current mid-sized problems require a nine-node cluster to maintain about 10^7 cells, on each of which 9 double precision numbers represent the electromagnetic fields and currents. A total number of 2×10^7 particles move across those cells, where each particle is characterized by 7 parameters. A typical job requires several days of execution time and produces more than 2 GB of output data. Recent developments of high-intensity pulses have created the demand to simulate much larger plasma volumes which would require execution times of several months. This motivates the thorough performance analysis of the existing code presented here.

We first give an overview over the PIC method and its implementation in RPP3D. The tools employed for code analysis are briefly discussed before a the analysis of single-node performance, load distribution, and communication is presented.

1.1 The PIC method

The propagation of a laser pulse through a plasma is governed by Maxwell's equations

$$\begin{aligned} \vec{\nabla} \times \ \vec{B} &= \frac{1}{c} \frac{\partial \vec{E}}{\partial t} + \frac{4\pi}{c} \vec{J} \\ \vec{\nabla} \times \ \vec{E} &= -\frac{1}{c} \frac{\partial \vec{B}}{\partial t} \ , \end{aligned} \tag{1}$$

where the three-component vectors $\vec{E}$ and $\vec{B}$ are electric and magnetic fields of the plasma. The derivatives are with respect to time t and the three spatial directions x, y, z. The currents $\vec{J}$ are generated by electrons that move according to the relativistic equations of motion

$$\frac{\partial \vec{p}}{\partial t} = q_e \left(\vec{E} + \vec{v} \times \vec{B} \right) , \text{ with } \ \vec{p} = \frac{m\vec{v}}{\sqrt{1 - \frac{v^2}{c^2}}}. \tag{2}$$

Here m and $\vec{v}$ denote mass and velocity of each electron and c is the velocity of light. A single electron contributes an amount of $q_e \vec{v}$ to the total current $\vec{J}$, where q_e is the electron charge. The contribution of ionic motion to the currents is described by an analogous equation with mass and charge adjusted accordingly

The main idea of PIC is to maximally exploit the locality of the above system of partial differential equations for an efficient implementation on a parallel computer. The total simulation volume is divided into a grid of cubic "cells", whose edges and surfaces carry the components of the local fields and currents in the respective directions (cf. Figures 1 and 2). The electrons are lumped together in "macro-particles" in the form of cubes of the same size as the cells, each representing a cloud of electrons. For simplicity, we do not distinguish in the following discussion between the macroparticles and the electrons.

Maxwell's equations are discretized using a first order finite difference scheme. As an illustration, the time-derivative of the x-component of $\vec{E}$ is approximated as

$$\frac{E_x^t - E_x^{t-1}}{\Delta t} = c \left(\frac{B_z(y) - B_z(y-1)}{\Delta y} - \frac{B_y(z) - B_y(z-1)}{\Delta z} \right) - 4\pi J_x \, . \tag{3}$$

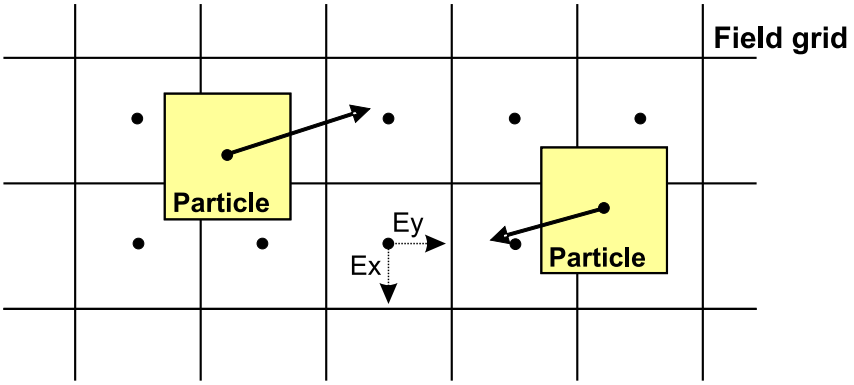

Figure 1: A two-dimensional illustration of the grid cells and two particles flowing continuously around. The center points of the cells and particles are also shown, as well as two field vectors are exemplified in one grid cell.

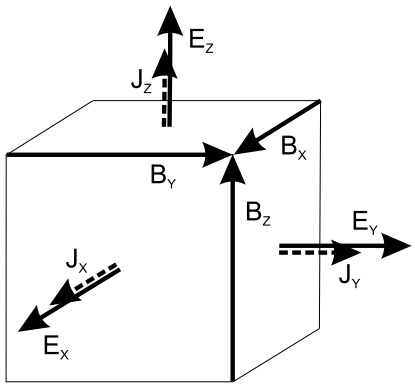

Figure 2: One cell of the grid and the nine values it maintains.

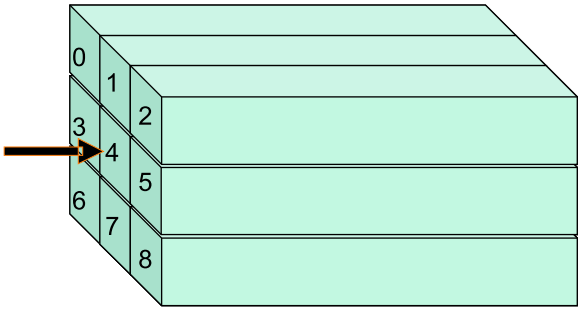

Figure 3: The separation of the simulation space in several subvolumes. Each subvolume is assigned to a cluster node. The arrow denotes the laser pulse which is applied at the center subvolume.

Particle motion, in turn, is continuous across the whole simulation volume. The equations of motion (2) are solved by a 4th order Runge Kutta algorithm. As field quantities are given only on discrete grid points, values for arbitrary particle coordinates are obtained by interpolation.

Electrical current is produced, when particles cross grid-cell boundaries. The contribution of a moving particle with total charge q_e to the surface component of $\vec{J}$ is weighted by the fraction of the particle cube that overlaps with the cell.

Memory consumption of the scheme above is determined by the need to store field and particle parameters of all cells and particles. RPP3D consumes a minimum of $X \times Y \times Z \times 9 \times [precision]$ Bytes of RAM to establish the simulation space. A typical mid-sized simulation space of $X = Y = 120$ and $Z = 800$ and using double precision floating numbers leads to a total consumption of approximately 850 MB from RAM. The actual numbers are somewhat larger because extra arrays for border values are used for performance reasons.

1.2 The Main Algorithm

The simulation volume is divided into subvolumes, which are distributed over parallel compute nodes as shown in Figure 3. Each time step consists of updates of electric and magnetic fields, particle motion, and updates of the cell currents. Between these steps one must communicate border values of the fields to neighboring compute nodes and transfer particles that move from one subvolume to the next.

Figure 4 shows a simplified flow chart of the RPP3D algorithm. The

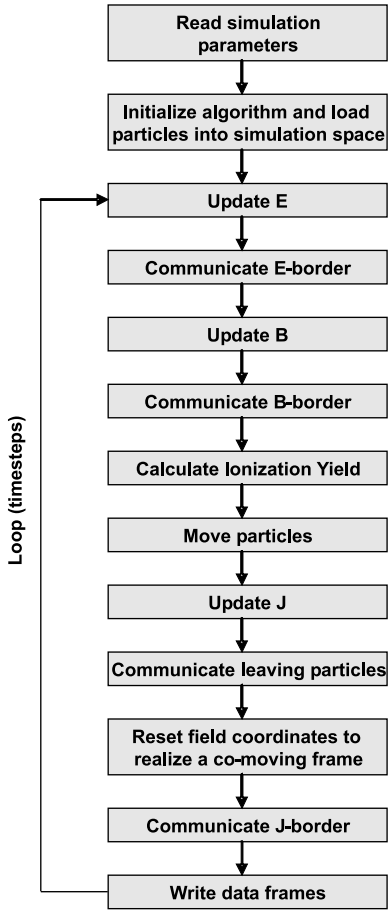

Figure 4: The main algorithm of the RPP3D code.

main loop starts immediately after the two initialization steps, which consists of parsing the simulation input file and creating the simulation space, i. e. the initial grid and particle quantities accordingly.

In the computation, a "co-moving frame" is used, i.e. the simulation volume itself propagates at the speed of light and the laser pulse can never leave it. In that way propagation over large distance can be simulated using only a limited volume. For the implementation this requires the shift of the coordinates of the field quantities after each time step.

The theoretical scaling behavior of that scheme is favorable as the communication is strictly nearest-neighbor. In principle, when the volume to surface ratio of the subvolumes is kept constant, linear scaling with the

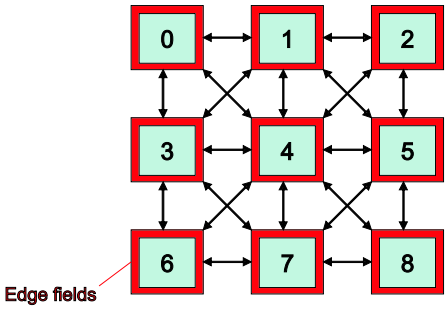

Figure 5: The field quantities of the border cells must be communicated to every adjacent cluster node.

number of compute notes could be expected (cf. Figure 5). In practice, an increase in volume is accompanied by changes in the physical parameters like laser field strength or density of particles which leads to an as yet unknown change in behavior of the equations (see discussion below).

1.3 Theoretical Code Analysis

The physical simulation volumes are measured in micrometers, but from a performance point of view, the cell count per dimension is more critical because each cell carries a fixed amount of electromagnetic field and current values. Therefore, larger simulation volumes correspond to an equally larger amount of cells. Obviously, the volume scales quadratically with the cross-section. That is, doubling both the X and Y dimension raises the lateral areas by two, but the volume—and also the computing load—is increased by four. Since there is no communication through the front and rear surfaces, the amount of field-related traffic scales proportionally with the cross-section (the X and Y dimension) or the Z dimension.

On the other hand, the particles-related traffic is more complicated and difficult to predict. Initially, at the start of the simulation, the particles are uniformly distributed. Since the simulated plasma is globally neutral, both electrons and ions must exist in equal numbers. That is the total number of particles is (typically) twice the number of cells which is predetermined by the volume.

Depending on the amount of energy injected by the laser pulse into the plasma, a more or less significant number of particles will leave the current subvolume of a computing node. But the pulse has also focusing capabilities, constricting certain particles in the current subvolume. In practice,

each computing node maintains an array of all particles that are present to this node. At every time step, leaving particles are communicated to the neighboring nodes, but this amount depends in a complicated way on the pulse geometry, the pulse intensity, the size of the simulation volume cross section, the boundary conditions, and other parameters.

1.4 Open Performance Questions

Although the algorithm and the source code are well known, due to the complexity of a >13,500 lines parallelized code, empirical studies of the performance and scaling behavior are needed. The following questions were still open:

- Do certain subroutines result in bottlenecks compared to other subroutines? Which code regions need to be optimized?
- What traffic volume is communicated between the compute nodes?
- Is the CPU or the network burden more critical?
- What amount of particle-related traffic is really generated?
- Does the computing and traffic load become non-uniformly distributed across the cluster nodes?
- Which scaling behavior can be expected for larger problem sizes?

The subsequent sections describe performance study in detail and present our main conclusions.

2 Runtime Performance and Communication Measurements

A variety of tools were employed for the analysis of node-performance and communication. The SCALEA package [1] gave a detailed node-differential analysis of the performance. For monitoring actual data rates close to hardware, the MRTG [2] tool was employed. Finally, for a detailed analysis of data flows, a new tool "WatchMPI" was developed. We give a brief description of these three tools below.

2.1 SCALEA

The SCALEA package and its instrumentation system allowed us to mark suspicious code regions by clamping them via special instrumentation function calls. Each of these "clamps" can be assigned a unique identifier, which refers to the corresponding performance metric measured, plus a code region designator, which allows to group code regions by type. Measurable performance metrics include wall-clock time, user time, system time, and others. See [1] for a detailed description.

The profiling tool also logs code region dependencies, that is, nested code regions can be easily tracked. This functionality supports a quick survey through the code and a subsequent examination in greater detail. Other than commonly used profiling tools, SCALEA allows to examine distributed cluster applications, that is, each single node is observed separately. The instrumentation library "SCALEA Instrumentation System" (SIS) supports C/C++ and Fortran programs, which are implemented based on OpenMP, MPI, or HPF+ routines.

We could successfully utilize SCALEA to measure various performance parameters for a number of code regions, such as the number of invocations and timing metrics. Figure 6 shows how the computing wall clock time is distributed over various interesting code regions per parallel instance. This way, certain critical code regions have been identified for recoding. In addition the results have been compared and verified by other profiling tools, in particular gprof and cachegrind, which could be applied on single-node runs only. Essentially the figures were the same except that the performance characteristics subtly change when the number of nodes is increased and the application becomes more and more distributed.

2.2 WatchMPI

A simple MPI traffic measurement tool was developed, named "WatchMPI", which allows to track each single data transaction between the computing nodes and even between single code regions which are automatically identified by a unique identifier. Network protocol issues such as TCP algorithms and timers are not taken into account. In order to improve our program architecture we are only interested in the traffic volumes passed to the MPI layer. In particular it is important to know how long each transaction takes. Subsequently we can easily determine the aggregate traffic rates injected into the network backbone to grasp any physical limitations.

WatchMPI consists of two components. First of all, a fast MPI wrapper

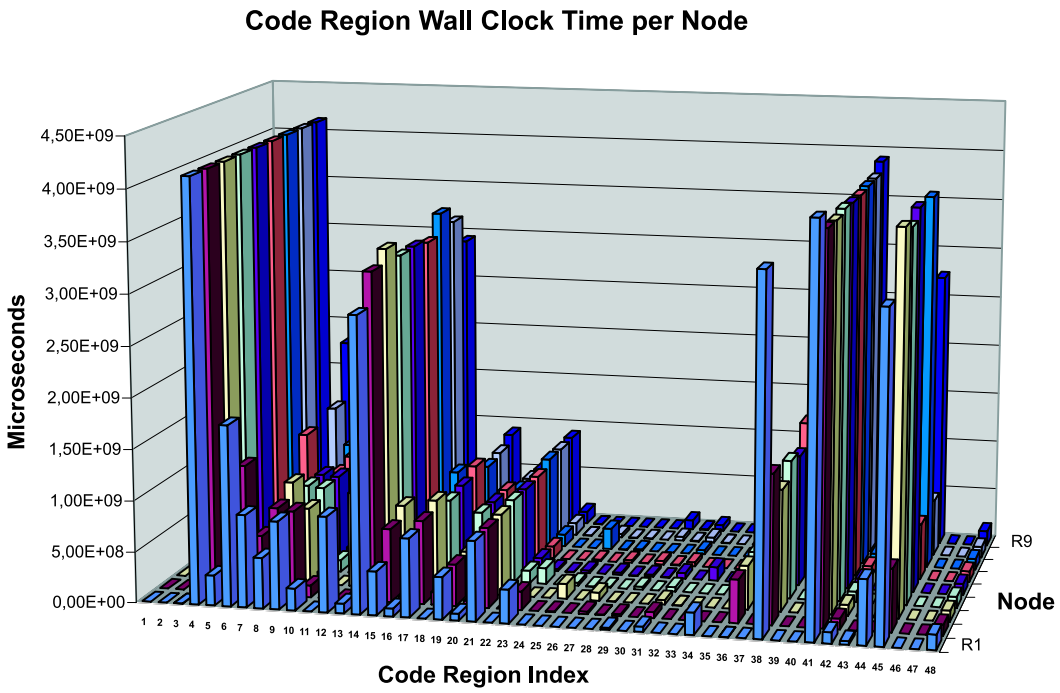

Figure 6: The computing wall clock time consumption across code regions and parallel instances. As it can be seen, most of the time is consumed by a few subroutines only.

written in C is used, which does the main work: it collects and logs statistical traffic data. The second component consists of a set of Octave modules to support the analysis of the logfiles. Octave [3] is a matrices-based calculation program similar to MatLab (but freeware) and nearly syntax equivalent, therefore these modules could be easily used with MatLab. Finally, a simple shell script transforms any Fortran or Fortran90 code automatically into a WatchMPI-instrumented version—that is, there is no need to mark each MPI-call separately.

After instrumentation, the program is invoked as usual. During the run, WatchMPI logs each transaction into a node-specific logfile. Each entry consists of caller-ID, begin- and end- timestamp, receiver-rank, and the number of transmitted payload bytes. Using this information one can easily determine which MPI-call of a specific node has sent how much data to another specific node, and also when an event occurred and how long it took.

Besides tools for data preparation (calibration) and meta-tools (tools that further enhance the operation) the following basic analysis tools are currently available:

- **rates** computes the aggregate data rate per specified time unit (seconds or even microseconds) and creates postscript plots automatically. Fine-grain traffic rate figures can be computed, as well as a total average. This tool allows to observe the total payload traffic rate injected into the physical network backbone. MPI saturation and network limits can be easily detected.

- **plapper** creates a table which lists all MPI calls of the program by ID and the corresponding traffic volume being sent by it. This is also useful for sanity checks of the program and to detect code regions that send unexpected huge (or low) volumes of data.

- **collapse** plots the aggregate data volumes injected into the network by superposing all transactions in wall clock time bins, that is, considering wall clock start and end times. This tool allows to analyze what data quantities are passed to the MPI layer at specific points in time.

- **twonodes** plots single data transactions versus time between each two nodes of the cluster. This tool allows to determine whether all nodes utilize the MPI interface equally or not.

- **txrate** plots the aggregate cumulative data volume versus time. That is, the slope (the first derivative) of this curve is the current aggregate

traffic rate. This tool provides a simple overview of the overall traffic characteristic of this run. For example, periodic patterns of the main loop or time-varying anomalies (features) such as initialization traffic can be easily determined.

The following example shall illustrate the usage of WatchMPI. For any given Fortran[1] program `myprog.f90` which utilizes MPI calls, the command `transmpi90.sh` automatically prepares (or "instruments") any F90 code in the current directory. As a second step the code must be build against the WatchMPI code and can be executed as usual. If nine cluster nodes were involved, WatchMPI will generate nine logfiles (watchmpi_0.log, watchmpi_1.log, ..., watchmpi_8.log), which contain the traffic measurement information. These logfiles can be analyzed using the Octave tools described above. The first command should always be `octave:1> timereset(9)`, which removes the offset from all stored timestamps. Note that each WatchMPI module requires the number of cluster nodes as an argument. For example `octave:2> rates(9,1)` will immediately generate a graph in Postscript format, containing a plot of the aggregate traffic rates, using one second as time base. Figure 10 is an example plot of this tool. Another example of a practical module is `octave:3> twonodes (9, 200)`, which creates Postscript plots that illustrate the traffic volume being sent from one node to another. If N nodes were specified, this module will produce $N(N-1)$ plots, in case every node has sent data to any other node. The second argument specifies how many observed transactions should be plotted, in this case no more than 200. In our investigations, the module `twonodes` was especially useful to analyze the network load distribution.

2.3 MRTG

As an immediate solution to analyze a per-node network traffic, we installed the Multi-Router Traffic Grapher (MRTG) at the front-end node of the cluster. MRTG [2] is a freeware tool which allows to read network interface card counters periodically and automatically plots traffic statistics in self-created HTML pages. MRTG was initially developed to measure IP traffic statistics of routers (hence the name) but it is a universal Simple Network Management Protocol (SNMP) client and can be also used to read the counters of Ethernet, Myrinet and Infiniband Switches.

[1] Currently, WatchMPI only supports Fortran 77 and Fortran 90. An extension for C/C++ is planned.

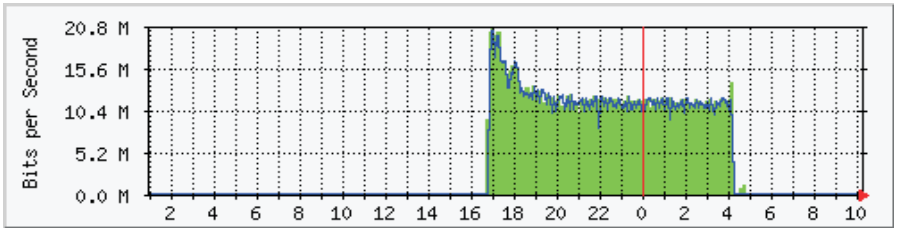

Figure 7: One example plot of MRTG illustrating the data rates of one cluster node during a 9-node RPP3D run.

Although MRTG turned out to be very practical to get a quick idea of the network load during a run, we needed a more precise tool, which allows to measure traffic rates with finer granularity since MRTG only supports time bases greater or equal five minutes. Furthermore, we needed to measure payload-based data rates, that is, without protocol overhead, in order to restrict the investigations solely on our program architecture. This lead to the development of WatchMPI.

3 Communication Analysis

As shown in figure 7, the traffic load of a single cluster node, which is processing a mid-sized problem lies between 10 and 20 Mbit/s. These values have been measured with MRTG and include all protocol overheads.[2] Therefore, the network backbone is currently no limit to our RPP3D application, assuming access rates greater or equal 1 Gb/s and a switch backplane of at least 1 Gb/s $\times$ number of nodes.

Interestingly, as shown in figure 8, the network backbone load (average aggregate data rate of the computing nodes) decreases when the simulation volume is increased, provided that a constant number of cluster nodes are involved. This observation has been made for numerous 4-node runs, as well as for 9-node runs.

Also it can be seen from figure 8, quadrupling the cross section (i. e. doubling the X and the Y dimension) reduces the average aggregate traffic rates by 60-90%. Obviously, the total traffic volume per loop cycle must be significantly increased because the surfaces of the subvolumes have been doubled and the number of particles even quadrupled.

[2]We assume full-sized Ethernet frames with 18 Bytes of header information plus additional 40 bytes for TCP/IP headers, leading to a protocol overhead of approximately 4 percent, disregarding any MPI overhead.

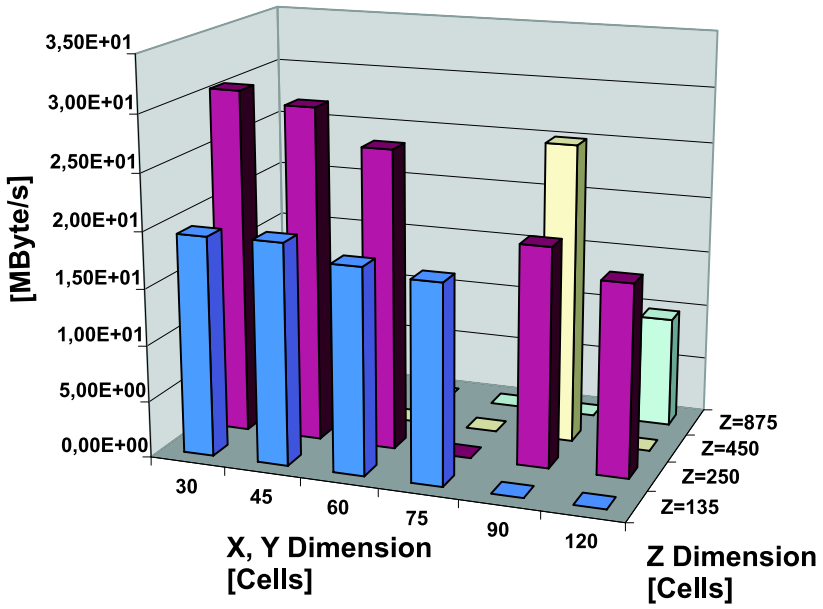

Figure 8: The aggregate average network load of 11 9-node runs versus simulation volume size. Note how the network load decreases when the simulation volume size is increased.

Table 1: The consumed wall clock time per main loop cycle in seconds (rounded) for four different dimensions of the simulation volume.

Volume Dimensions	Seconds/cycle
30×30×250	0.1
60×60×250	0.3
120×120×250	1
120×120×875	3

An empirical comparison of the "instantaneous traffic rate" has been made using a "collapse" algorithm, which aggregates and superposes all transmitted traffic volumes of all nodes in wall clock time bins. This WatchMPI feature does not provide a traffic rate based on arbitrary time bases, which would always smoothen extremely short bursts, rather it illustrates what amount of traffic load is passed to the MPI layer at any instant of time. For example the 9-node 60×60×250 run showed bursts up to 1,4 Mbytes "per instant", while the 9-node 120×120×250 run reached 3,5 Mbytes per instant. That is, the empirical scaling factor for the traffic bursts during a single pass of the main loop is approximately 2.5.

This empirical scaling behavior leads us to the assumption that the ratio between CPU load and network load is increased when the problem size is increased. Doubling the X and Y dimension quadruples the cross section as well as the volume of each subvolume. That is, the number of cells and the number of field quantities used for the calculations, are also increased by a factor of four. As table 1 shows, these assumptions are indeed supported by empirical observations, as the processing time per loop cycle is actually increased by a factor of 3 or 4.

Watchmpi labels each MPI-call with an unique index, which allows to draw the transmitted data volume versus "code region". We denote a code region any arbitrary consistent chunk of code (typically a subroutine) which passes data to the MPI interface. Figure 9 shows the traffic statistics of a typical example job, with dimensions of 60×60×250. Clearly, the field-related traffic is about 3-4 orders of magnitude larger than the particle-related traffic. In the example, the particle-related traffic has a magnitude of 10-100 MByte compared to 200-1000 MByte for the field-related traffic.

Another interesting observation is the initial negative slope of the average network backbone load at the beginning of a job. As it can be seen in

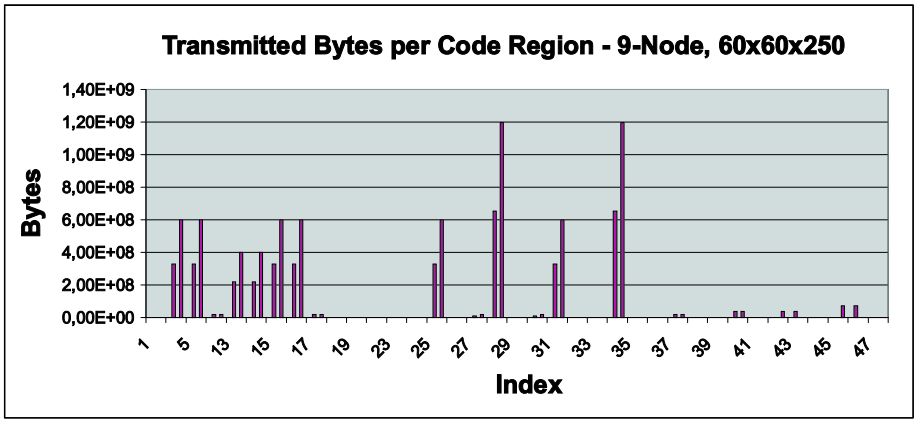

Figure 9: Total traffic volumes transmitted per code region. Code regions with traffic volumes below $2 \cdot 10^8$ communicate only particle information. All other code regions shown here communicate field-related information.

figure 10, the traffic rate is nearly twice as high when the job is started and reaches a constant average value after approximately 6000 seconds. This observation has been made with arbitrary problem sizes and the explanation is as follows: At the moment the laser pulse hits the front surface of the simulation volume, all cells have zero values for the field quantities and the calculation of the Maxwell equations is immediately finished. During the propagation of the laser pulse, more and more cells are affected by the distortion of the electromagnetic field and hence the calculation load increases linearly. Finally, when the laser pulse hits the rear surface of the simulation volume, the CPU load cannot be further increased and the observed traffic rate reaches the saturated value. At this moment, any further propagation of the laser pulse is implemented by a coordinate-shift of the field coordinates, hereby establishing a "co-moving frame".

It should be mentioned, that (currently) the particle-related communication is more efficiently implemented than the field-related communication. While particles are maintained in a dynamically managed array and it is obvious to only communicate those particles which are really leaving the volume, field values of any border-cell are always communicated—even if the assigned field quantities have not changed. As soon as traffic reduction becomes a critical issue, an improvement of the field-related data structures must be considered.

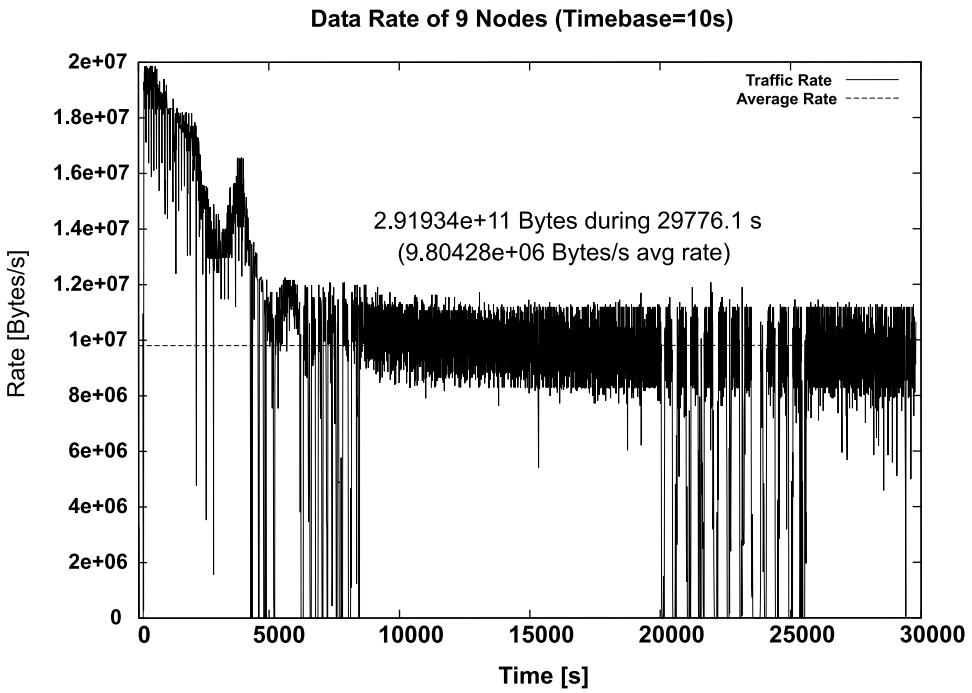

Figure 10: The data rate evolution of a typical 9-node run. The initial negative slope corresponds to inflow of the electromagnetic Al fields of the laser pulse.

4 Conclusion

SCALEA and WatchMPI turned out to be practically usable tools to investigate the performance and bottlenecks of Beowulf applications. Using them in combination, the most critical questions could be answered and also several optimization approaches could be planned. In particular:

- Certain subroutines which constitute critical bottlenecks have been identified.
- Currently, the aggregate traffic volume communicated between the cluster nodes does not hit the limits of a Gigabit/s network backbone[3]
- The ratio between the CPU load and the network load increases with an increased problem size. Typically, the scaling behavior drops the network load by approximately 30% when the simulation volume is quadrupled.
- The particle-related traffic is almost negligible compared to field- related traffic. Although the number of particles is twice as high as the total number of cells, only a small amount of them crosses the borders of a subvolume per time step.
- Both SCALEA and WatchMPI revealed that the computing and traffic load roughly remains uniformly distributed across the cluster nodes.
- The network load will only be limited by the network backbone if the basic algorithm will be significantly accelerated. This issue must be considered during the ongoing code optimizations.

Additional interesting analysis approaches could be performed because both SCALEA and WatchMPI provide various different types of measurement data, which can be correlated and might reveal further insights from a parallelized code. It should be stated that analysis tools for parallelized programs should be as flexible and automatic as possible. Typically, a single "performance experiment" takes a lot of preparation time, including instrumentation of the code, compilation time, and input file preparation, as well as a significant post processing time needed to analyze the data. According

[3]As been told by the cluster administrators, Jumbo frames, i. e. oversized frames greater than 8 KB, are currently not implemented. Since RPP3D generates typical message sizes between 40 and 800 KByte, the use of Jumbo frames would further increase the overall network throughput.

to our experience, the automation of the whole process is still an important issue.

All described performance tests have been made under "production conditions", that is, also other jobs were frequently running in parallel and consumed a more or less significant fraction of the network backplane bandwidth. Nevertheless, the described behavior has been repeatedly observed and can be considered reproducible.

References

[1] Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Parallel Programs. *Concurrency and Computation: Practice and Experience*, 15(11-12):1001–1025, 2003.

[2] Tobias Oetiker. The multi-router traffic grapher. *http://people.ee.ethz.ch/ oetiker/webtools/mrtg/*.

[3] John W. Eaton. Gnu octave. *http://www.octave.org/*.