A Petri-Net-Based Approach to Modeling Communication Algorithms for HPC Molecular Dynamics Simulations

Theresa Werner[1], Christof Päßler[2], Martin Richter[1], Ivo Kabadshow[2] and Matthias Werner[1]

[1]Chemnitz University of Technology, Straße der Nationen 62, Chemnitz, 09131, Germany
[2]Jülich Supercomputing Centre, Wilhelm-Johnen-Straße, Jülich, 52428, Germany

Abstract
With the development of increasing node numbers in High Performance Computing (HPC), i.e. strong-scaling, we get the problem that many HPC applications become communication-bound. Thus, there is a need for optimizing communication. Apart from improving the communication protocols, one can work on problem-specific communication patterns. A commonly used, efficient communication pattern for the simulation of homogeneously distributed particle systems is the *Shift* communication. Aiming to use the Shift for our own molecular dynamics simulation library, FMSolvr, we modeled it with timed Petri nets in order to get an estimation of the communication time. For the time analysis we use Max-Plus Algebra. We put our model's timing predictions up against an implementation of the Shift and found that our model is correct. However, our approach is over-modeling the problem.

Keywords
timed Petri nets, (max,+) algebra, distributed systems, communication modeling, molecular dynamics simulation

1. Introduction

Particle simulations are of interest in many fields of science, e.g. plasma physics, polymer science, solid state physics, and biophysics. But it is and always was a very time-consuming matter, because for each simulation timestep all forces that might influence a particle must be calculated for all particles. This means that one must handle a computational complexity of $\mathcal{O}(N^2)$ for each simulation timestep. With systems of millions or billions of particles, this leads to unreasonable runtimes in order to simulate a useful time span. Major improvements regarding the computational complexity can be achieved with fast summation methods like the Fast Multipole Method (FMM). They can reduce the computational complexity to $\mathcal{O}(N \log N)$ or even $\mathcal{O}(N)$. The FMM, which is used for our Molecular Dynamics Simulation (MDS) library FMSolvr[1], achieves a complexity of $\mathcal{O}(N)$. For the FMM the simulation space is split into

PNSE'23: International Workshop on Petri Nets and Software Engineering, June 26 – 27, 2023, Lissabon, Portugal
*Corresponding author.
thewe@hrz.tu-chemnitz.de (T. Werner); i.kabadshow@fz-juelich.de (I. Kabadshow)
www.researchgate.net/profile/Theresa-Werner-5 (T. Werner)

[1]code at fmsolvr.org

small cubic boxes (spatial decomposition with an octree[2]). With distributing inter-dependent sub-regions of the simulation space over different nodes, the question of efficient data exchange arises.

At first, the time cost for computation in High Performance Computing (HPC) was much higher than for communication. Simulation was computation-bound. But processors became faster, calculation cheaper, and HPC workstations grew into HPC clusters, until today we have the problem that strong-scaling leads to communication-boundness regarding inter-node communication. Thus, communication is a topic of optimization again. There are developments in the field of communication protocols such as LCI [1] or GASNet [2], but we are also interested in optimizing communication by using problem-specific communication patterns. Hence, we did a Systematic Literature Review (SLR) on what had already been done when it comes to communication in MDS [3], and we found that there is only one communication scheme that is used throughout all works: *Shift* communication [4]. It is a very simple but elegant communication algorithm that achieves a communication complexity (with regard to the number of messages) of $\mathcal{O}(k)$, where k is the range of our near-field interactions in units of box length (more on this in Sec. 2.2).

Since the Shift appears to be dominant in this field of research, we wanted to examine if it is suitable for FMSolvr. Given certain system properties and constraints, we modeled the Shift upon those properties using Timed Petri Nets (TdPN)[3]. Petri nets appear to be a good choice because other than for example CSP (Communicating Sequential Processes) they can easily be evaluated regarding timing behavior with the help of Max-Plus Algebra (MPA, also (max,+) algebra). Having a formal model of the Shift will allow us to compare the Shift to other communication schemes. One such comparison that is of special interest to us, is the Team Shift developed by Driscoll, Geroganas, and Koanantakool [6].

As for the structure of this work: Section 2 will explain the mechanics behind MDS and the meaning of the terms used in the introduction such as near-field, Section 3 will introduce the Shift communication, and Section 4 will walk the reader through the necessary theory of TdPNs and MPA. Then, Section 5 explains how we modeled the Shift based on our system properties and constraints, and Section 6 will compare the model predictions with the measurements from the implementation. Finally, Section 7 will give a discussion of the methods and results as well as an outlook to our future work.

2. Molecular Dynamics Simulation

This section will introduce the reader to the basics of MDS: how to distribute the particles over the computation nodes in Section 2.1, and the specifics of near-field interactions, which we are focusing on with this work, in Section 2.2.

[2]Octree means that we split the simulation space into eight big boxes, then we split those big boxes into eight smaller boxes, which we split again into eight smaller boxes, and so on, until we have our required granularity of splitting up the simulation space.

[3]We choose the abbreviation TdPN because TPN stands for Time Petri Nets, which are a different type of time-dependent nets [5].

2.1. Load Distribution

In MDS there are three ways to distribute particles over nodes: atom, force, and spatial decomposition [4]. Of the three, FMSolvr uses spatial decomposition. The great advantage of spatial decomposition compared to its two alternatives is that for range-limited interactions (see next section) communication only needs to happen between computation nodes that hold neighboring regions, so we have a communication complexity of $\mathcal{O}(k)$ with $k \ll P$, where P is the total number of nodes. Compared to this, atom and force decomposition require data of all other nodes (atom: $\mathcal{O}(\log P)$) or at least a considerable subset of them (force: $\mathcal{O}(\log \sqrt{P})$)[4].

Spatial decomposition means that we take our simulation space, divide it into small boxes, and distribute these boxes over our computation nodes. Of course, there are different ways to divide the space. When one handles sparse, clustered particle systems, one would like to use methods similar to Barnes-Hut [7], which divides the space into boxes of different sizes in order to achieve good load balancing[5]. But if one needs to handle homogeneously distributed particle systems, then one can divide the space into same-sized cubes (edge length $b_x = b_y = b_z = b$). This is what we will be looking at.

2.2. Defining the Near-field

When a particle is moving among other particles, it is subject to and the cause of Coulomb forces (aka. electrostatic forces). The Coulomb force is strong at a short range but its influence decreases with growing distance ($\sim 1/d^2$), which means that when two particles move apart, the individual influence that one particle has on another slowly melts into the background noise of other particles' force fields. The distance at which the individual particle can no longer be distinguished from the background is the end of the near-field, this distance is called the cut-off radius R_{cut}.

In order to supply for each particle the data of particles within its cut-off radius, a node must import the data of particles, which surround its box within the distance of R_{cut}. Figure 1 shows the import area in relation to the cut-off radius. The node holding the dark-blue box must communicate with every node that holds a (partly) light-blue box. This means that there is no difference in the number of communication partners between Figure 1a and 1b. Only with a radius bigger than the box length, the number of communication partners changes.

In order to determine the number of communication partners of a node, we use $k = \lceil R_{\text{cut}}/b \rceil$ to calculate the number of boxes covered by the import area in one direction. For Figures 1a and 1b k is one, and for Figure 1c k is two. Then the total number of communication partners cp of one node in a three-dimensional simulation is defined by

$$cp = (2k+1)^3 - 1 \tag{1}$$

[4]For more details on atom and force decomposition as well as their communication complexity, please refer to [4].
[5]An alternative to Barnes-Hut would be a sparse octree.

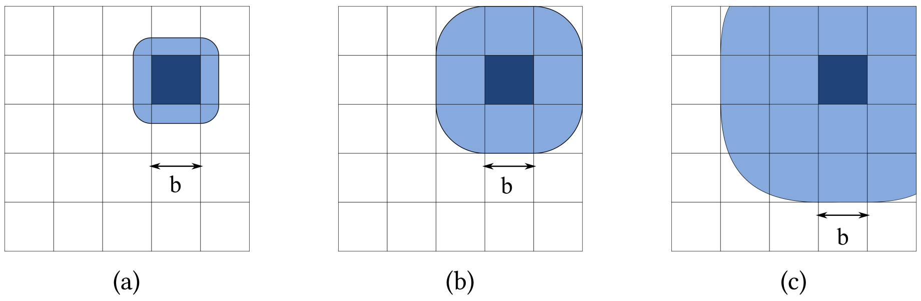

Figure 1: The three version of the cut-off radius [3]: dark blue is the target box, light blue is the import area of that box depending on the cut-off radius. (a) $R_{\text{cut}} < b$, (b) $R_{\text{cut}} = b$, (c) $R_{\text{cut}} > b$ with b being the edge length of a box.

3. Communication in MDS – the Shift

With a growing k, the number of communication partners increases quickly. In order to ensure an efficient exchange of data, one needs an efficient communication algorithm. Here is where Shift communication enters the game. A predecessor of the Shift is well-known in parallel programming [8, Ch.4.2], the MDS version was introduced by [4] in 1993. In three communication phases the particle data is distributed to all the places where it is needed while a node only needs to communicate with its direct six neighbors: Left and Right, Front and Back, Up and Down. Figure 2 depicts the process (for simplicity the terms ’box’ and ’node’ are used interchangeably):

Figure 2a serves to give the reader an understanding of the surroundings of a box. The whole simulation space looks like a grid of boxes where this 3×3 grid is a sub-grid of the whole. To describe the Shift, we want to focus on the blue and yellow box, so the rest of the grid is made invisible (Fig. 2b). We will use $k = 1$ for the description.

In the first step of the Shift the node exchanges its data with its direct neighbors Left and Right (Fig. 2c). At the end of this step the node holds all the row data within $k = 1$. For the second step it combines the Left and Right data as well as its own into one message and exchanges it for the package of its Front and Back neighbors (Fig. 2d). Now it hold all the knowledge about the plane. In the third step of the Shift the accumulated data of the plane is exchanged for the plane data of the Up and Down neighbors (Fig. 2e). So, in order to get all the required data for $k = 1$, a node must only exchange six messages with its direct six neighbors.

In case of $k > 1$, the neighbors help daisy-chaining the data further down the row. This means, the Right neighbor would hand the node the data of the second neighbor to the right and the Left neighbor of the second left neighbor. The data of the third neighbor to the right would pass through the second neighbor to the right and the Right neighbor before it arrives at the target node. So, a node must exchange a total of $6k$ packages to receive all the required data (this means $12k$ messages in total). However, the reader might have noticed that a node must wait for specific data to arrive before it can either pass it on or start the next phase. This is where time dependency between nodes comes into play.

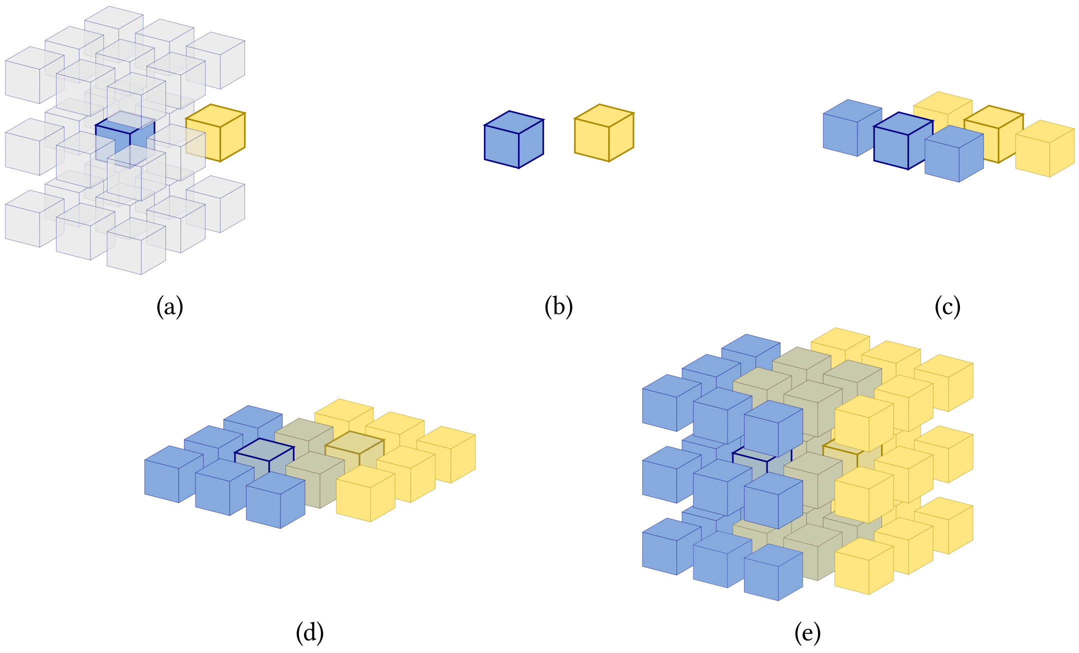

Figure 2: Shift communication: The data of the blue/yellow box is first distributed along the row (c). Then, the accumulated data of the row is sent along the column (d). And last, the accumulated data of the plane is sent along the towers (d). If every node follows this pattern, all nodes receive all the data they require within $6k$ exchanges ($12k$ messages).

4. Timed Petri Nets and the Max-Plus Algebra

In order to capture the time dependencies between nodes, Petri nets with timed transitions can be used. A timed transition here represents a message passing process. This section will do a brief recapture of Timed Petri Nets (TdPNs) and explain how to transfer TdPNs into the mathematically evaluable form of a max-plus matrix.

4.1. Capturing Time with a Net

A marked Petri net is characterized by a five-tuple $\mathcal{N} = (P, T, E, W, m_0)$ where P and T are places and transitions respectively, E are the edges connecting a place with a transition or a transition with a place, W are the multiplicity (weights) assigned to edges, and m_0 is the initial marking. If we assign each transition $t \in T$ a duration $d \in D$, we gain a TdPN, which is a six-tuple of $\mathcal{D} = (P, T, E, W, m_0, D)$ where D is called the duration function[6]. [5]

To make a practical example we shall look at a node that wants to send data to another node. The sending process can be captured by a timed transition, while the places will represent certain internal states of the node. In this case, we have two states: 'before sending' and 'done sending'. Figure 3a depicts the net with its initial marking, and Figure 3b shows the marking after the node has finished sending. The time that passed between state one and state two is

[6]In our case D is not a function but simple time values for each transaction.

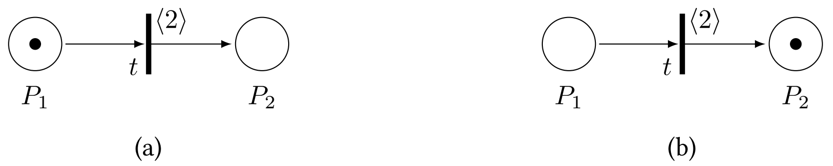

Figure 3: TdPN of a sending process: (a) shows the inital marking before sending, and (b) shows the marking after sending. P_1 represents the node's internal state of 'before sending' and P_2 the state of 'done sending'. The transition t, which represents the sending, takes the time 2.

the time assigned to the transition t, denoted by the number in angle brackets.

4.2. The Max-Plus Algebra

The Max-Plus Algebra (MPA) comprises of two operations: the maximum operation $\oplus$ and the addition $\otimes$. They are defined as follows:

$$a \oplus b = b \oplus a = \max\{a, b\}$$

$$a \otimes b = b \otimes a = a + b \ .$$

The addition ($\otimes$) has priority over the maximum ($\oplus$) if not ordered otherwise by brackets. MPA also knows a neutral element for the maximum, the zero element $\varepsilon = -\infty$, and a neutral element for the addition, the identity element $\eta = 0$, for which holds:

$$\varepsilon \oplus x = x$$

$$\eta \otimes x = x \ .$$

Equivalent to the sum $\sum$ and the product $\prod$ in linear algebra, MPA defines an accumulative maximum and sum:

$$\bigoplus_{i=0}^{n} c_i = \max\{c_0, c_1, ..., c_n\}$$

$$\bigotimes_{i=0}^{n} d_i = d_0 + d_1 + ... + d_n \ .$$

And just like in linear algebra, MPA can be applied to matrices. Taking two matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ of size $n \times m$, the maximum is an element-wise maximum defined as

$$A \oplus B = [a_{ij} \oplus b_{ij}] \ .$$

Taking a matrix $A = [a_{ij}]$ of size $n \times m$ and a matrix $B = [b_{ij}]$ of size $m \times n$, the MPA matrix plus operation is defined as

$$A \otimes B = \left[\bigoplus_{k=1}^{m} a_{ik} \otimes b_{kj} \right] .$$

We also find a neutral element for the MPA matrix plus operation, the identity matrix I, which is defined as

$$I = \begin{bmatrix} \eta & \varepsilon & \cdots & \varepsilon \\ \varepsilon & \eta & & \vdots \\ \vdots & & \ddots & \vdots \\ \varepsilon & \cdots & \cdots & \eta \end{bmatrix}.$$

In regard to vectors, the MPA matrix-vector addition ($\otimes$) has an important role in analyzing Petri nets. For a matrix A of size $n \times m$ and a vector $\vec{x}$ of length m, it is defined as

$$A \otimes \vec{x} = \begin{bmatrix} \bigoplus_{k=1}^{m} a_{ik} \otimes x_k \end{bmatrix}.$$

For more detailed information or as a reference for the next section, please find [9] or [10].

4.3. Max-Plus and Timed Petri Nets

MPA is a tool to transform TdPN into a mathematically evaluable matrix form. For the transformation we use a matrix A of size $|P| \times |P|$ ($|P|$ is the total number of places in the net), and if there is a transition that connects place p_x with place p_y, the matrix element A_{yx} gets the time value d that the transition takes, otherwise it gets the value $\varepsilon = -\infty$ [11].

Let us extend the example of a node sending a message. Now, we have two nodes that want to send each other a message. Let us assume they cannot send and receive at the same time, then one must send after the other. If one node wants to send, it needs the "ok" of the other node, so in order for the transition to fire, both nodes must provide a token: the sending node to signal that it has data ready, the receiving node to signal that it is ready to receive that data. The respective Petri net is shown in Figure 4a. Place P_0 is the internal state of node n_0 that means 'data ready', and P_1 is n_0's state 'done sending'. Place P_2 is the internal state of node n_1 that means 'ready to receive' while P_3 is n_1's state 'done receiving'. Equivalently, the sending process of node n_1 is shown in Figure 4b. When we combine the two nets into one TdPN, we obtain Figure 4c, the net of a classical ping-pong message exchange. One can see that the two nodes have timing dependencies.

Our final Petri net has six places, so our connection matrix A is of size 6×6. When we fill in the connections between places like described in the first paragraph of this section and assume that $d_i = 1$, we get the connection matrix

$$A = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ d_0 & \varepsilon & \varepsilon & d_0 & \varepsilon & \varepsilon \\ \varepsilon & d_4 & \varepsilon & \varepsilon & d_4 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ d_0 & \varepsilon & \varepsilon & d_0 & \varepsilon & \varepsilon \\ \varepsilon & d_4 & \varepsilon & \varepsilon & d_4 & \varepsilon \end{bmatrix} = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 1 & \varepsilon & \varepsilon & 1 & \varepsilon & \varepsilon \\ \varepsilon & 1 & \varepsilon & \varepsilon & 1 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 1 & \varepsilon & \varepsilon & 1 & \varepsilon & \varepsilon \\ \varepsilon & 1 & \varepsilon & \varepsilon & 1 & \varepsilon \end{bmatrix}.$$

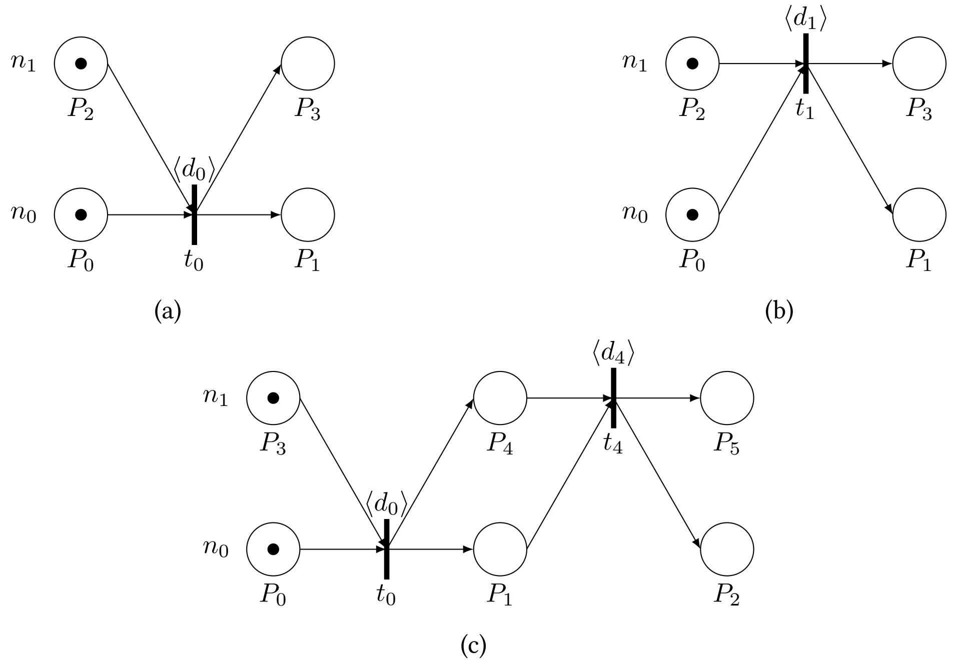

Figure 4: TdPN of a ping-pong message exchange between two nodes: (a) shows the net of node n_0 sending and n_1 receiving, (b) shows the net of node n_1 sending and n_0 receiving, and (c) combines the two individual TdPNs to the net of a ping-pong message exchange. The transitions are named after the place of the sending node; transition t_i takes time d_i.

What we would like to find now is the longest path. This can be achieved by MPA adding ($\otimes$) the connection matrix with itself. A^l is able to return the longest path of length l (with $l \in \mathbb{N}$) and is defined as

$$A^l = A \otimes A \otimes ... \otimes A = \bigotimes_{i=1}^{l} A \ .$$

But we want the overall longest path, which is returned by the star matrix A^* defined as

$$A^* = I \oplus A \oplus A^2 \oplus ... \oplus A^m$$

where $m \geq |P| - 1$, because this guaranties that all possible connections are included even in the case of a node ring. Now we can MPA add ($\otimes$) our input vector to the star matrix and get as the result the times when the places of the net are reached. Let us assume that node n_0 starts sending immediately and node n_1 is immediately ready to receive, then our input vector is a zero vector $\vec{v} = \vec{0}$. By performing MPA matrix-vector addition $A^* \otimes \vec{v}$ we get an output vector

$\vec{u}$ that tells us what state is reached after what time[7]:

$$A^* \otimes \vec{v} = \begin{bmatrix} \eta & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 1 & \eta & \varepsilon & 1 & \varepsilon & \varepsilon \\ 2 & 1 & \eta & 2 & 1 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \eta & \varepsilon & \varepsilon \\ 1 & \varepsilon & \varepsilon & 1 & \eta & \varepsilon \\ 2 & 1 & \varepsilon & 2 & 1 & \eta \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 0 \\ 1 \\ 2 \end{bmatrix} = \vec{u}\,.$$

Vector $\vec{u}$ shows that the states represented by place P_0 and P_3 are immediately reached (due to our assumption that both nodes are ready at time 0), that the states that are represented by place P_1 and P_4 ('n_0 done sending' and 'n_1 done receiving') are reached after one time unit, and that the states represented by place P_2 and P_5 ('n_0 done receiving' and 'n_1 done sending') are reached after two time units. If we compare this to our TdPN in Figure 4c, the reader will find that the numbers are congruent with the paths through the net.

5. Modeling the Shift

In this section we will give a detailed description of how we modeled the Shift on the basis of our system properties. Therefore, we will summarize our system properties and additional constraints and then show the Petri net model for the first dimension (row) and how to extend it to the second and third dimension (column and tower).

5.1. System Properties

The communication on the JURECA-DC cluster is location invariant[8]. This means that it does not matter if two communication nodes are placed in the same rack or on different racks. The physical distance of nodes has a neglectable impact on the latency of the exchanged message.

For our implementation, we constrain the system by demanding that sender and receiver must have a handshake for the transfer. For small messages below 256 kB[9], MPI does not demand a handshake but uses the eager protocol (see OpenMPI eager [12]). This means a Send call returns immediately because the sender knows that the receiver has enough buffer to receive the message without explicit agreement to reception. We do not want this behavior (for now), because it causes different timing behavior on the sending and receiving side. Instead we demand a transfer where both partners must shake hands, a rendezvous. To achieve this we used `MPI_Ssend`, which forces a rendezvous for all message sizes.

[7]A special case are non-deterministic Petri nets. In that case the output vector delivers the worst case times to reach a place.

[8]We avoid the more correct term of place invariance because it might cause confusion due to a different meaning in the context of Petri nets.

[9]We measured the MPI eager threshold on our cluster and it contradicts the commonly found value of 12 kB. We do not know why, but our measurements show a clear jump at 256 kB.

5.2. The Petri Net for One Dimension

First, we will look at the Petri net for a one-dimensional Shift communication. As we already introduced with the examples in Section 4, each node is represented by a number of places that stand for different states which the node reaches with progressing time (see example: ready to send $\rightarrow$ done sending = ready to receive $\rightarrow$ done receiving). In order to send a message, a sending node needs the agreement of the receiving node that the latter is ready to receive data. Together they can enable the transition that represents the sending process. The firing of the transition takes a certain time and produces a token for the sending node, which means that the sending is finished, and a token for the receiving node, which means that the data is received.

In order to avoid non-determinism (for now), we dictate a communication pattern: the nodes can either start with all transactions to the right and then switch to finish the transactions to the left or vice versa. Figure 5 shows the first pattern for $k = 1$. Places, which are in a horizontal line, represent the states of one node n_i. The transactions are indexed with the number of the preceding place of the sending node; the durations have been omitted since we have location invariance, i.e. all transitions take the same time. For the nodes n_2 and n_3 the presented section of the whole net is sufficient for a MPA analysis of the time dependencies for $k = 1$. The number of nodes required for the analysis is increasing quickly with a growing k. It can be put into a formula in the following way[10]:

$$n_{\text{net}} = 8k - 2\ .$$

5.3. Multiple Dimensions

For the other two phases (column and towers), the nets look identical to the first dimension, only the transition times change due to higher message load. One net of the column dimension requires n_{net} neighboring nets from the row dimension. Equivalently, one net of the tower dimension will need just as many neighboring column TdPNs. The different phases can be handled independently with the output of the previous phase being the input for the next one. For this work we limited our analysis to the first phase to see whether the model can reflect the actual time behavior.

6. Model Meets Reality

In this section we will present the results of the theoretical analysis with MPA and the actual measurements from the implementation.

For the MPA analysis we used Python to generate and analyze the matrices. The input for the transition durations is the average message passing time between nodes of the cluster because we wanted to see whether measuring the average latency and feeding it into the matrix would be enough to gain an accurate prediction for the communication time. With an OpenMPI ping-pong message (using `MPI_Ssend`) we measured the average message passing time between nodes using 1,000 ping-pongs. The results of the ping-pong measurements for different message sizes are listed in Table 4 in the Appendix.

[10]The formula results from analyzing the nets for $k \in [1, 5]$.

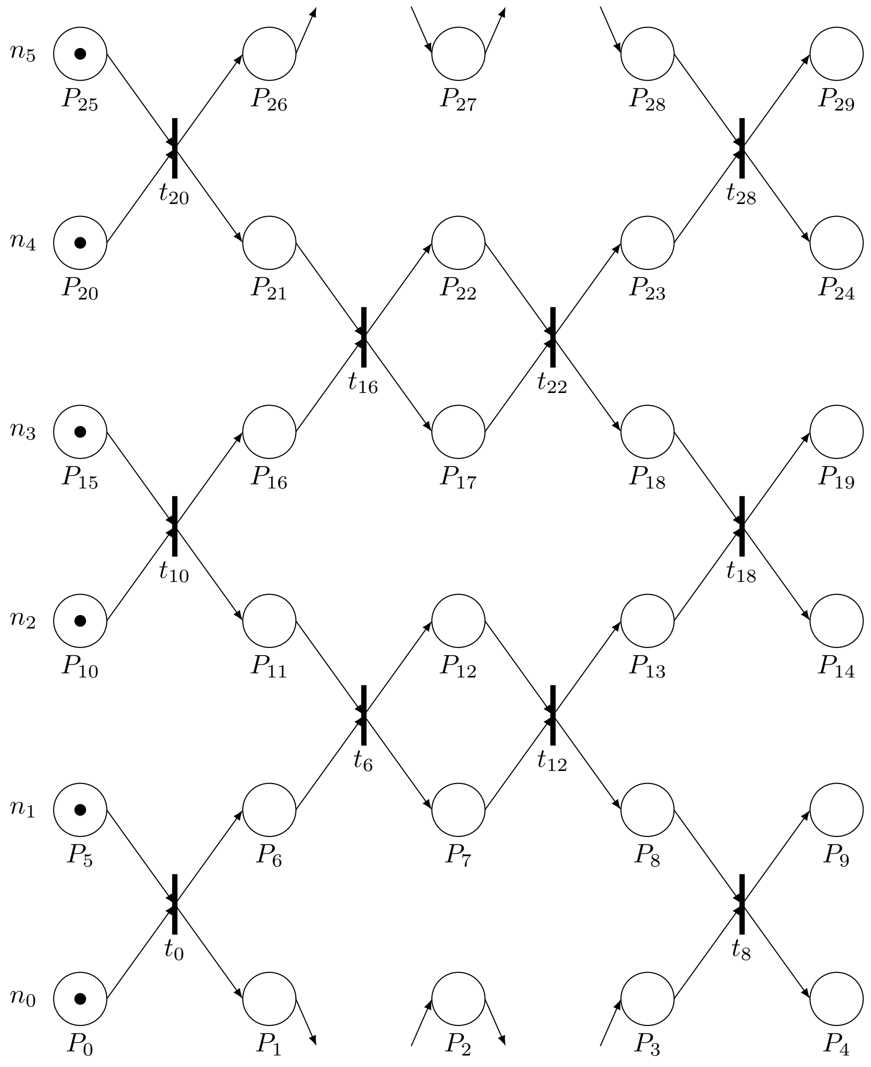

Figure 5: Section of the TdPN that is relevant in order to evaluate the dependencies of the Shift communication in the first dimension (along the row, $k = 1$). The transitions are named after the place of the sending node; all transitions take the same time due to location invariance of the cluster. The net continues on the top and bottom end.

The implementation of the Shift was done with OpenMPI 4.1.2 (GCC 11.2.0) on the JURECA-DC cluster (InfiniBand network). We used `MPI_Ssend` for all send routines in order to force a rendezvous for each message. The amount of part-taking nodes P appears to be neglectable (see Tables 5-7 in Appendix), so we will focus on the measurements performed with 64 nodes to get the full spectrum of k-values.

The results from the MPA analysis compared to the implementation measurements can be found in Tables 1-3. The predictions are within the standard deviation σ of the measurements, so the model can be considered correct for the range of k-values that we used. However, we

Table 1
MPA analysis vs. implementation measurements for 10 bytes load.

k	measured latency [ns]	σ [ns]	predicted MPA [ns]
1	11,360	4,332	9,216
2	19,478	1,765	18,432
3	30,071	8,554	27,648
4	39,338	11,080	36,864
5	48,424	10,598	46,080
6	58,701	16,412	55,296
7	66,476	9,267	64,512
8	75,920	10,316	73,728
9	88,762	24,108	82,944
10	97,072	25,594	92,160

Table 2
MPA analysis vs. implementation measurements for 1,000 bytes load.

k	measured latency [ns]	σ [ns]	predicted MPA [ns]
1	13,804	5,294	12,184
2	22,493	3,715	24,368
3	34,814	6,499	36,552
4	45,422	8,145	48,736
5	55,956	11,090	60,920
6	68,121	20,231	73,104
7	78,428	15,781	85,288
8	93,193	21,057	97,472
9	102,292	18,898	109,656
10	114,215	20,178	121,840

Table 3
MPA analysis vs. implementation measurements for 100,000 bytes load.

k	measured latency [ns]	σ [ns]	predicted MPA [ns]
1	62,708	7,353	59,772
2	127,668	19,557	119,544
3	191,647	29,054	179,316
4	263,956	65,629	239,088
5	319,701	65,497	298,860
6	467,980	492,450	358,632
7	448,284	83,874	418,404
8	565,866	389,992	478,176
9	585,111	109,965	537,948
10	671,048	169,343	597,720

can also see that the prediction is more accurate for message sizes of 10 and 1,000 bytes than for 100,000 bytes.

7. A Word at the End

This section is the conclusion of our work and discusses our approach to modeling, our model, our measurements, and our result. Last, it gives an outlook on what we want to do next.

7.1. Discussion

7.1.1. Measurements

We noticed that the predictions are less accurate for the message size of 100,000 bytes. Our thought is that for messages of this size we can see network saturation effects which we do not have when we perform a single ping pong measurement. A way to improve the model here would be to create network noise while measuring the ping pong.

Another thing we noticed it that if we limit the ping pong measurements and the Shift measurements to the 20% smallest values (the best cases), the prediction becomes stunningly accurate (differences of a few hundred nanoseconds). However, we cannot name a reason for the 20% threshold, so this needs further evaluation.

7.1.2. Model

Our model can be considered correct. However, the MPA matrices are growing exponentially with the k-value, which makes them time consuming to evaluate. Moreover, we are over-modeling our system. Our model incorporates more detail than what is necessary for the analysis. The ability to assign different times to all transitions in order to model slower and faster nodes is not necessary due to location invariance of the cluster. Here, we reduced the model's complexity to a simple calculation of average message latency multiplied by the number of message exchanges/number of rendezvous.

By using a duration function that is dependent on the bandwidth and traffic in the network instead of duration values, a model like ours could be helpful to analyze the influence of network congestion. It would also be more suitable for analyzing the timing behavior that results from the lack of location invariance, i.e. different duration values per node pair.

We chose Petri nets in the first place because of their ability to include time easily and handle non-determinism, which we will need in the future once we switch to an implementation with `MPI_Isend`. However, TdPNs do not seem like a suitable approach for our problem or our cluster due to the reasons mentioned above.

7.2. Future Work

For the presented analysis, we forced MPI not to use the eager protocol but the rendezvous protocol, which is increasing the latency. A future model will incorporate the eager protocol. Also, non-determinism in the form of e.g. `MPI_Isend` will be included since it might be able to reduce communication time by introducing flexibility in the case where a communication partner is not instantly available.

Acknowledgments

We thank Andreas Beckmann for assistance with the implementation of the one-dimensional Shift. This work is supported by DFG project FMhub, project Nr. 443189148.

References

[1] H.-V. Dang, M. Snir, LCI: Low-level communication interface for asynchronous distributed-memory task model, in: ACM Conference (Conference'17), 2018.

[2] D. Bonachea, P. Hargrove, GASNet-EX: A High-Performance, Portable Communication Library for Exascale, in: Proceedings of Languages and Compilers for Parallel Computing (LCPC'18), 2018. doi:10.25344/S4QP4W.

[3] T. Werner, I. Kabadshow, M. Werner, Systematic literature review of data exchange strategies for range-limited particle interactions, in: Proceedings of the 12th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, 2022, pp. 218–225.

[4] S. Plimpton, Fast Parallel Algorithms for Short-Range Molecular Dynamics, Journal of Computational Physics 117 (1995) 1–19.

[5] L. Popova-Zeugmann, Time petri nets, Berlin, Heidelberg: Springer, 2013.

[6] M. Driscoll, E. Georganas, P. Koanantakool, E. Solomonik, K. Yelick, A Communication-Optimal N-Body Algorithm for Direct Interactions, in: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, 2013, pp. 1075–1084.

[7] J. Barnes, P. Hut, A hierarchical O(N log N) force-calculation algorithm, Nature 324 (1986) 446–449.

[8] A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, Second Edition, Addison-Wesley Professional, 2003.

[9] F. Baccelli, G. Cohen, G. J. Olsder, J.-P. Quadrat, Synchronization and Linearity, An Algebra for Discrete Event Systems, 1992.

[10] T. Rambow, U. Kiencke, Max-Plus-Algebra, Teil 1, Automatisierungstechnik 53 (2005). doi:10.1524/auto.2005.53.10_2005.A1.

[11] T. Rambow, U. Kiencke, Max-Plus-Algebra, Teil 2, Automatisierungstechnik 53 (2005). doi:10.1524/auto.2005.53.11.a5.

[12] FAQ: Tuning the run-time characteristics of MPI shared memory communications, 2023. URL: https://www.open-mpi.org/faq/?category=sm.

A. Additional Tables

Table 4
Average message passing time in relation to message load.

Load [byte]	Latency [ns]	Load [byte]	Latency [ns]
0	2138	1,000	3046
5	2137	5,000	4002
10	2304	10,000	4925
50	2843	50,000	10883
100	2864	100,000	14943
250	2890	250,000	27267
260	3018	260,000	28489
500	2937	500,000	47305

Table 5
MPA analysis vs. implementation measurements for 10 bytes load.

k	$P = 64$ [ns]	$P = 32$ [ns]	$P = 16$ [ns]	$P = 8$ [ns]	$P = 4$ [ns]	MPA [ns]
1	11360	11018	10896	10841	10830	9216
2	19478	19574	19464	19449	19458	18432
3	30071	29551	29312	29212	-	27648
4	39338	38851	38554	38438	-	36864
5	48424	48565	48140	-	-	46080
6	58701	58706	58066	-	-	55296
7	66476	66580	66072	-	-	64512
8	75920	76233	75631	-	-	73728
9	88762	87499	-	-	-	82944
10	97072	96654	-	-	-	92160

Table 6
MPA analysis vs. implementation measurements for 1,000 bytes load.

k	$P = 64$ [ns]	$P = 32$ [ns]	$P = 16$ [ns]	$P = 8$ [ns]	$P = 4$ [ns]	MPA [ns]
1	13804	13322	13219	13107	13068	12184
2	22493	22681	22659	22685	22676	24368
3	34814	34783	34515	34483	-	36552
4	45422	45164	44848	44778	-	48736
5	55956	56301	55993	-	-	60920
6	68121	68140	67347	-	-	73104
7	78428	78255	77623	-	-	85288
8	93193	92577	91433	-	-	97472
9	102292	102080	-	-	-	109656
10	114215	113395	-	-	-	121840

Table 7
MPA analysis vs. implementation measurements for 100,000 bytes load.

k	$P = 64$ [ns]	$P = 32$ [ns]	$P = 16$ [ns]	$P = 8$ [ns]	$P = 4$ [ns]	MPA [ns]
1	62708	63796	64679	64538	64387	59772
2	127668	128520	132292	131663	131256	119544
3	191647	191821	191611	191115	-	179316
4	263956	260298	260356	259143	-	239088
5	319701	320535	321957	-	-	298860
6	467980	439503	435429	-	-	358632
7	448284	451280	452199	-	-	418404
8	565866	548509	550349	-	-	478176
9	585111	583438	-	-	-	537948
10	671048	665027	-	-	-	597720