INTERNATIONAL JOURNAL OF COMPUTERS
DOI: 10.46300/9108.2021.15.5 Volume 15, 2021

Evaluating the Mesh-connected Multiprocessor Running a Parallel Algorithm Representing the Gravitational Interaction Between N Bodies

Jamil Al-Azzeh,
Al-Balqa Applied University, Jordan

Received: December 21, 2020. Revised: March 9, 2021. Accepted: March 16, 2021. Published: March 22, 2021.

Abstract: **In this study, a parallel algorithm to solve the problem of gravitational interaction between N bodies is considered. The peak performance of an 8×8 mesh-connected multiprocessor system while implementing the algorithm is evaluated. Further, the mappings of parallel threads of the algorithm onto the multiprocessor are studied based on two criteria for minimizing the communication delay. The corresponding real application performance of the multiprocessor is estimated.**

Keywords: **mesh-connected multiprocessors, performance, thread mapping, communication latency, minimax criterion, minimaximin criterion.**

I. INTRODUCTION

Currently, the development of multiprocessor-based systems is considered a priority in the evolution of computer technology (1). Emerging computational problems that cannot be solved within practical and affordable time using traditional single-processor systems have contributed to the extensive development and implementation of multi-core and many-core multiprocessor systems (4) that are expected to drastically enhance the computer performance (8,10,12,26).

Multiprocessor systems have been applied in various contexts to solve computationally complex problems (14,9). For example, such systems have been used in various physics domains, and high-performance systems are required by the aerospace and automotive industries and nuclear power engineering. Parallel computing systems are the primary solution basis in financial and economic forecasting, oil and gas exploration problems, weather and climate forecasting, traffic flow optimization in megacities, as well as biological research and genetics. In addition, multiprocessor-based systems are used in safety-critical real-time systems (13).

II. HARDWIRED VERSUS RECONFIGURABLE MULTIPROCESSOR SYSTEMS

Conventionally, multiprocessor systems employ hardwired or reconfigurable architecture. Hardwired architectures are fixed, i.e., they cannot be modified to satisfy the requirements of a given problem. Typically, such systems involve a set of conventional processor units and standard communication tools, i.e., networking hardware and protocols. The mesh-connected multiprocessor-on-a-chip is an example of such systems. In contrast, reconfigurable systems can adapt to a given algorithm to provide the best-fit architecture. Typically, such systems are based on dedicated hardware, e.g., application-specific integrated circuits (ASIC) or programmable logic devices (PLD). When compared to reconfigurable architectures, hardwired architecture exhibits several significant disadvantages such as relatively low inter-processor data exchange speed and limited communication network bandwidth. Consequently, hardwired systems can reach peak performance only while running "loosely coupled" parallel applications that do not produce heavy inter-processor traffic. Sets of threads in separate processor cores may need to be synchronized

frequently if the implemented algorithm is poorly decomposed and allocated, thereby contributing to performance degradation (3,23,24,22,15,19,17,5,11,18,2). To increase the efficiency of the hardwired parallel computing in a general case, specific methods must be used to map the parallel applications or solution algorithms onto the system such that the communication cost is minimized (20). The mapping procedure should be performed according to a certain criterion.

III. MULTIPROCESSOR SYSTEM-ON-A-CHIP

Tilera and Adapteva processors are typical examples of hardwired multiprocessor systems-on-chips. Tilera's recently released Tile-Gx64 is a general purpose 64-bit RISC-based multiprocessors targeted at servers running large multithreaded applications (21). The block diagram of the Tile-Gx36 multiprocessor is shown in Figure 1.

Tile-Gx processors are manufactured according to 40 nm microelectronic technology. The Tile-Gx36 has 36 processor cores connected via a dedicated iMesh network. The Tile-Gx36 operates at a clock speed of 1.2 GHz, and its power consumption does not exceed 24 W. Each core of the Tile-Gx36 chip has a first and second-level cache memory unit, several memory controllers, and multiple I/O controllers. Each core can run a standalone operating system (OS), and multiple cores can be managed by a single multiprocessor OS. Tile-Gx-based systems have nearly perfect scalability. Processor cores can be grouped to reach maximum performance for a given application. The Tile-Gx36 provides up to 40 Gbps of bandwidth for network communication.

The Tile-Gx processor family reduces system development costs by providing built-in memory and I/O controllers, thereby reducing the number of external components that need to be designed and implemented. The TileDirect technology facilitates consistent I/O operation directly to processor node cache, which significantly reduces delays in processing data packets. Tilera's distributed coherent cache (DDC) system increases the performance of multithreaded and shared memory applications. Processor cores can be grouped into independent compute clusters running separate applications. The multiprocessor is programmed using C and C++ languages, which allows developers to port existing applications to them.

The Tile-Gx functional core is a 64-bit VLIW processor that executes 64-bit-wide instructions. This core processor has 64 registers in a register file, a three-level pipeline with up to three instructions per cycle, a doubled first-level cache for 32 KB data and 32 KB instructions, and a 256 KB second-level cache. The clock frequency is in the range 1-1.5 GHz. The third level cache is formed as a union of the caches located in individual processor cores, Tilera's so called DDC.

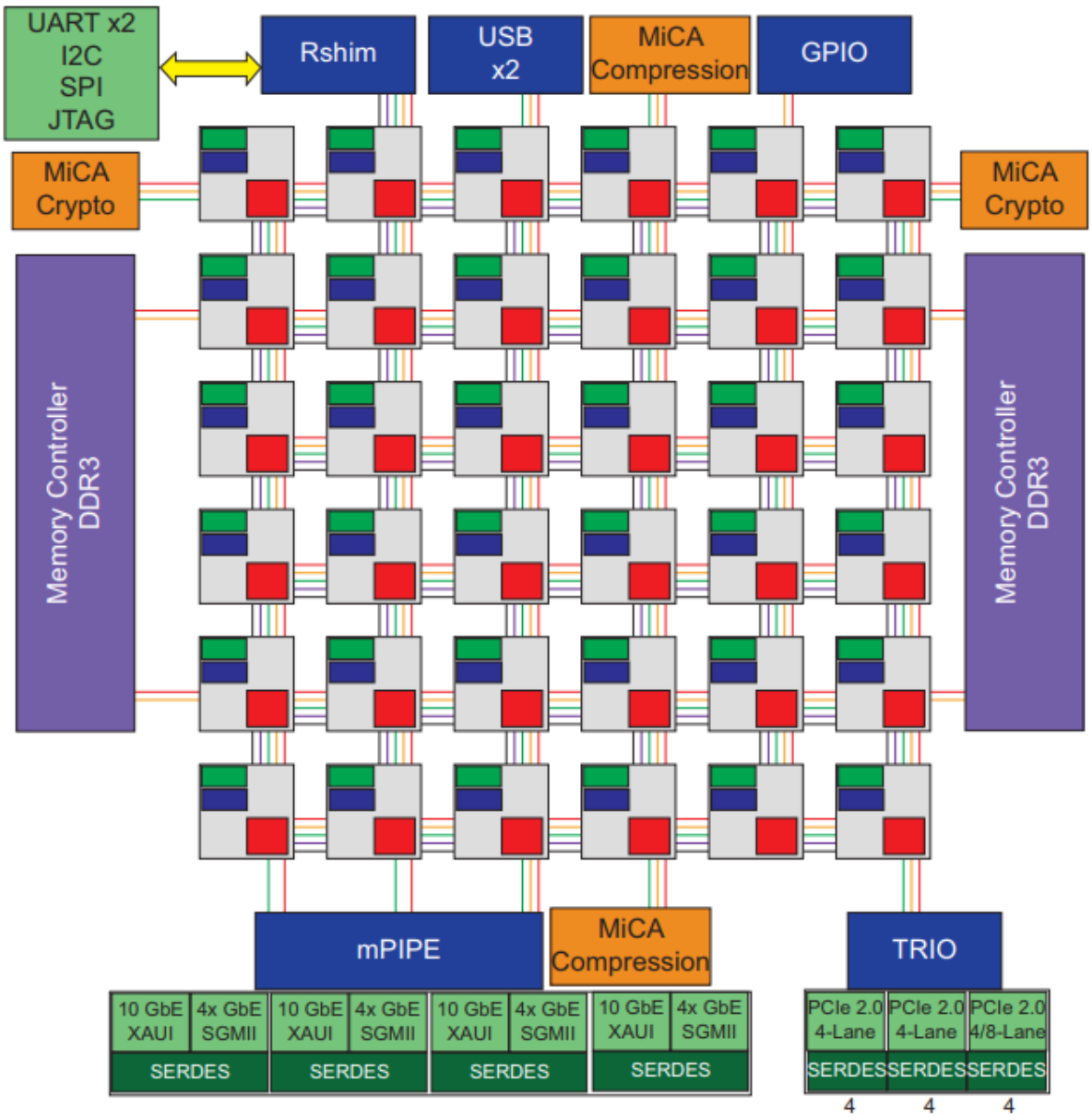

Figure 1. Tile-Gx multiprocessor block diagram

The inter-core data transfer speed of the Tile-Gx chip is approximately 200 Tb/s. Four memory controllers (DDR3) provide memory bus bandwidth of 500 Gb/s. The multiprocessor can have up to eight 10 Gigabit Ethernet interfaces (XAUI), up to three second generation PCI-e interfaces, up to 32 Gb Ethernet interfaces (MAC), an mPIPE network packet processing device, as well as encryption and data compression hardware. Thus, the processor can run network applications with a traffic intensity of 40-80 Gb/s (with packet processing bandwidth of 80 Gb/s and VPN traffic processing bandwidth of 40 Gb/s). Note that PCI-E bandwidth reaches up to 80 Gb/s. The total power consumption of the chip is 10-50 W. Tile-Gx processor topology is typical for multiprocessors and is suitable for solving problems characterized by independent object or data parallelism. Such system organization, i.e., multiple instruction/multiple data, requires a host

computer generating a stream of instructions and a large number of processor nodes operating in parallel and processing particular data streams. Thus, the peak performance of the system is simply the sum of the processor nodes' peak performances across the mesh. However, to solve a wide range of practical problems and increase system efficiency, communication between processor cores must be organized such that the computational capabilities of a multiprocessor system are maximized (7,6,16,25).

IV. EFFECTS OF THREADS-TO-PROCESSORS MAPPING ON MULTIPROCESSOR PERFORMANCE

Threads-to-processors mapping significant affects communication delay in mesh-connected multiprocessors like Tile-Gx. When determining thread mapping prior to uploading threads to the cores, a delay estimation method to investigate the effectiveness of possible threads-to-processors mappings must be formulated. A minimaximin criterion has been proposed for such estimation purposes. This criterion calculated the minimum communication delay for each pair of processor nodes corresponding to the selection of the least loaded path between two processors in subsequent routing. After determining the minimum delays for all pairs of processors in the multiprocessor, the maximum value is chosen. Note that the maximum delay limits the inter-processor data exchange time. This calculation method estimates the worst case of possible inter-processor data stream overlap, the delay does not increase after routing more than the value obtained by this method. However, a minimax communication delay criterion is used. This minimax criterion characterizes data exchange time for a given pair of processors without considering inter-processor data stream overlaps occurring during simultaneous data exchanges for a given set of processor pairs. The imposition of data routes increases the volume of inter-processor communication and drastically reduces multiprocessor performance when solving tightly coupled tasks.

In this paper, we focus on employing the minimaximin criterion when mapping parallel threads to processor nodes and demonstrate that the resulting communication latency reduction leads to increased multiprocessor performance. In this context, we consider a very tightly coupled task as an example because communication delay minimization is for crucial for such problems. Our example task is the gravitational interaction of N bodies, which is a well-known classical celestial mechanics and gravitational dynamics problem first formulated by Isaac Newton.

V. FORMULATION OF EXAMPLE PROBLEM

The target problem is described as follows. In a void, there exist N material points (bodies) with masses m_i, initial positions $r_{i0} = r_i\big|_{t=0}$, and velocities $v_{i0} = v_i\big|_{t=0}$. Here, pairwise interaction of points is controlled by the universal gravitation law, and the gravitational forces are assumed to be additive. The positions of the points for all subsequent moments in time must be determined.

The numerical solution to this problem can be found by calculating the increments for the velocities and coordinates using a time step of Δt as follows.

$$\vec{v}^{(t)} = \vec{v}^{(t-1)} + \vec{a}^{(t)}\Delta t,$$
$$\vec{x}^{(t)} = \vec{x}^{(t-1)} + \vec{v}^{(t)}\Delta t$$

Here, $\vec{v}^{(t)}$, $\vec{v}^{(t-1)}$, $\vec{x}^{(t)}$, $\vec{x}^{(t-1)}$, and $\vec{a}^{(t)}$ denote the velocity of the target point at time t, the velocity of the target point at time $t-1$, the position (coordinate) of the target point at time t, the position (coordinate) of the target point at time $t-1$, and the acceleration of the target point at time t as calculated according to Newton's second law:

$$\vec{a}^{(t)} = \frac{\vec{F}}{m},$$

where $\vec{F}$ is the resultant force affecting the body calculated as the sum of the attraction forces between the current body and all other bodies. The attraction force between the *i*th and *j*th bodies is determined as follows according to the universal gravitation law:

$$\vec{F}_{ij} = G\frac{m_i m_j}{r_{ij}^3}\vec{r}_{ij},$$

where *G* is the gravitational constant. Here, r_{ij} is the distance between the *i*th and *j*th bodies and is calculated as follows:

$$r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2},$$

Where x_i, x_j, y_i, y_j, z_i, and z_j denote the corresponding coordinates of the *i*th and *j*th bodies in three-dimensional space.

VI. SOLUTION TO EXAMPLE PROBLEM

Based on the above formal statements, the algorithm used to calculate the coordinates of an interacting body comprises the following steps. Note that the number of required necessary operations is given in parentheses.

1) Determine the distance between a body and all of its peers (9 FLOPs)

2) Calculate the attraction force between the given body and all other bodies (4 FLOPs)

3) Estimate the resulting force affecting a body

4) Determine the acceleration of the given body (1 FLOP)

5) Compute the velocity of the body at moment *t* (2 FLOPs)

6) Calculate the coordinates of the given body at time *t* (6 FLOPs)

In Figure 2, a parallel-sequential solution to the considered problem is given with a set of threads (shown as circles).

It is possible to solve the stated problem in parallel according to the decomposition of Figure 2 in a Tile-Gx-like multiprocessor (Figure 1), where each processor core performs a loop to calculate the coordinates of the given body, transmits the result to all peers calculating the same for the other bodies, and then proceeds to the next iteration. As shown in Figure 2, the presented parallel algorithm is fully connected, and the weights of arcs represent the amount of data required to store the coordinate values in memory. Note that the accuracy of the calculations is important; therefore, it is necessary to use the double-precision floating-point format, which requires 8 bytes of memory per number. Here, it is necessary to transfer three coordinates for a given body; thus, the arcs of the graph have a weight of 24.

VII. EVALUATING MULTIPROCESSOR PERFORMANCE WHEN SOLVING TARGET PROBLEM

Based on the above example, the real performance of an 8×8 Tile processor running the algorithm shown in Figure 2 can be evaluated using the previously mentioned thread mapping technique, which enables the minimization of inter-processor communication delay. Here, we introduce the following dedicated multiprocessor performance indicator:

$$P = \frac{V_{\text{comp}}}{T_{\text{comp}} + T_{\text{comm}}},$$

where V_{comp} is the calculation amount (FLOPs), T_{comp} is calculation duration in seconds, and T_{comm} is data transfer time, which is estimated as the maximum across the set of processor pairs and depends on the given threads-to-processors mapping.

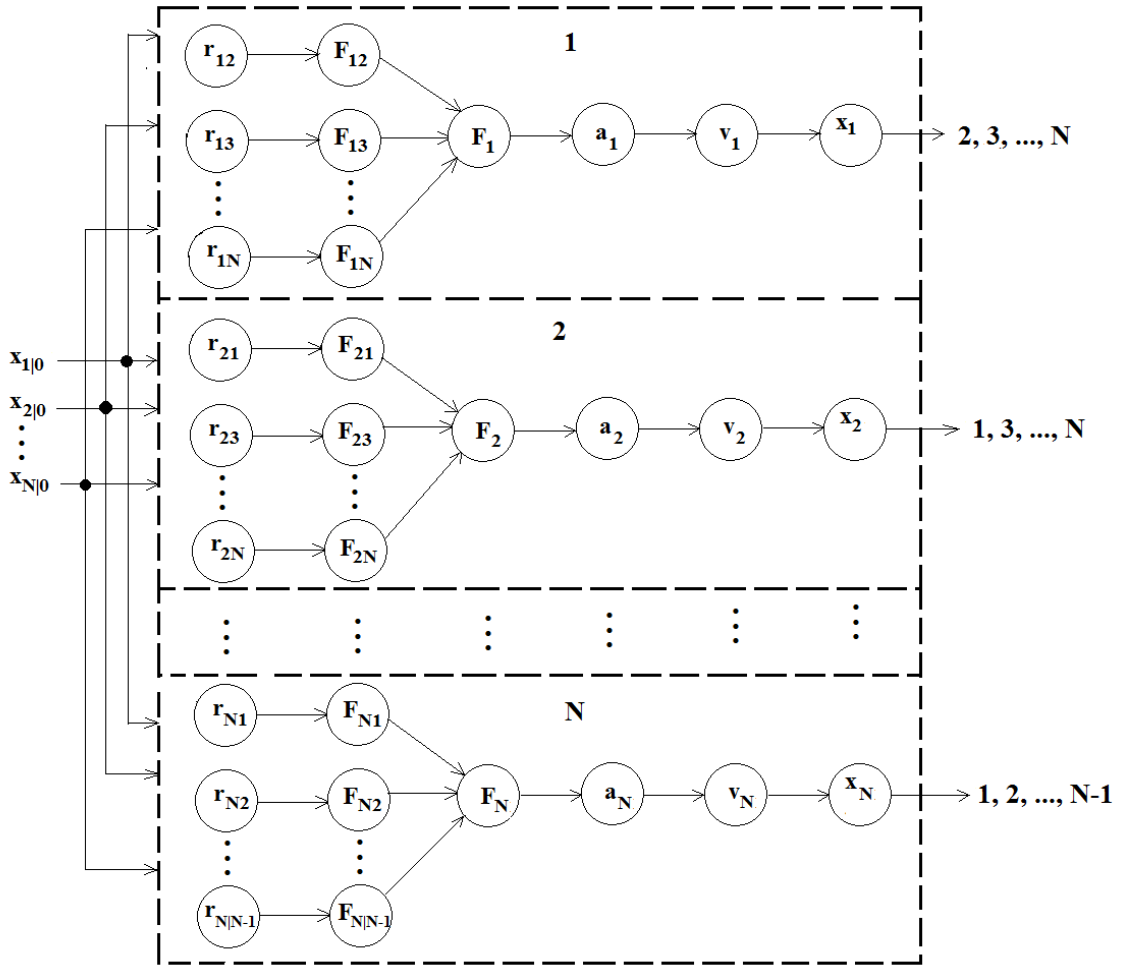

Figure 2. Parallel-sequential solution to *N* interacting bodies problem

The calculation amount in the body coordinate estimation algorithm is determined as follows:

$$V_{\text{comp}} = 9(N-1)N + 4(N-1)N + (N-1)N + N + 2N + 6N = \\ = 9N^2 - 9N + 4N^2 - 4N + N^2 - N + 9N = 14N^2 - 5N,$$

where *N* is the number of bodies.

The multiprocessor is a homogeneous computing system in which each processor core runs a calculation loop for a single body; thus, compute time is determined by multiplying the execution time of a single loop of the coordinate calculation by the number of iterations of the algorithm. According to the characteristics of the target Tile processor, the maximum core frequency is 1 GHz: thus, the execution time of the coordinate calculation period in nanoseconds is equal to the number of clock cycles for the floating-point operations that make up the loop:

$$T_{\text{comp}} = \frac{(N-1)c_1 + (N-1)(c_2 + c_3) + c_4 + c_5 + 3c_6}{10^9},$$

where $c_1 = 54$ clock cycles are required to calculate the distance between bodies, $c_2 + c_3 = 116$ clock cycles are required to estimate the attraction force plus the resulting force, $c_4 = 80$ clock cycles are required to obtain the acceleration of a body, $c_5 = 14$ clock cycles are required to calculate the velocity of a given body, and $c_6 = 14$ clock cycles are required to estimate the coordinates of a body.

Assuming all data exchanges are performed simultaneously, the inter-processor data transfer time is limited by the worst-case pairwise delay resulting from a threads-to-processors mapping produced by the mapping algorithm. In a serial byte-by-byte data exchange, the data transfer time for a pair of processors (excluding overlapping data routes) is determined as the product of $m \cdot d \cdot c$, where *m* is the number of bytes transferred, *d* is the number of consecutive links between a given pair of processors, and *c* is a constant factor equal to the transfer time of 1 byte per link. Note that the minimaximin criterion considers possible route overlaps that require summation of data transmission times; therefore, T_{comm} is defined as the maximum data transfer time for any pair of processors calculated by the minimaximin criterion using $c = 10$ ns, which corresponds to the capabilities of the communication subsystem of modern multiprocessors on a chip.

In the following evaluation, we assume a one-quarter multiprocessor load, which means that the 16-body problem is solved and only 16 (of 64) processor cores are utilized. In the worst case, the multiprocessor is fully loaded (i.e., a 64-body problem) and the threads-to-processors mapping optimization cannot reduce communication delay because any permutation on the set of 64 threads yields the same total amount of data to be transferred between pairs of processor cores.

Using 16 processor nodes, the peak performance of the multiprocessor can be determined if T_{comm} is assumed to tend to 0. Here, $V_{\text{comp}} = 3504$ FLOPs and $T_{\text{comp}} = 2686$ ns: thus, we obtain the following.

$$P = \frac{V_{\text{comp}}}{T_{\text{comp}}} = 1.3 \text{ GFLOP/s}$$

The threads-to-processors mapping algorithm was tested previously using the minimaximin (1) and minimax (4) criteria to evaluate which criterion provides deeper optimization. Figure 3(a) shows the given initial mapping of 16 threads to calculate the coordinates of bodies. Here, the maximum distance between corresponding processors is six hops. Note that this mapping is optimal relative to the maximum distance between processors; however, it yields a significant increase in communication delay when routing along the shortest paths due to the full connectivity of the data exchange graph and significant overlaps among data transmission channels. In Figure 3(a), all shortest paths between the allocated processors lie within the dashed rectangle.

The following simulation results were obtained. The maximum data exchange time for any pair of processors for the initial mapping shown in Figure 3(a) according to the minimax criterion without considering channel overlap is 1920 ns. In contrast, with multiple route overlaps, the worst data exchange time is 13440 ns, which time corresponds to the pair of processors 1-28. The only shortest path between two processors is located on the same horizontal or vertical line of the multiprocessor mesh, which enables the estimation the minimum number of route overlaps on any shortest path under the previously discussed routing condition. Consider the shortest path between processor cores 1 and 28, i.e., 1-2-3-4-12-20-28. For this case, shortest channels (selecting only the shortest path) are shown in Table 1.

Table 1
Channel length calculation depending on the position of processor pairs

Processors pairs in same horizontal/vertical line	Channel length	Processors pairs in different horizontal/vertical lines	Channel length
1-2	1	1-12	4
1-3	2	1-20	5
1-4	3	2-12	3
2-3	1	2-20	4
2-4	2	2-28	5
3-4	1	3-12	2
4-12	1	3-20	3
4-20	2	3-28	4
4-28	3		
12-20	1		
12-28	2		
20-28	1		

According the data given in Table 1, we can calculate the minimum possible delay of this shortest path in consideration of the unidirectional data exchange between processors 1 and 28, the shortest distance between which is 6. The minimum possible delay of this path is obtained as follows.

T_{SUM} = (24·1·6+24·2·4+24·3·2+24·6·1)·10 = 24·10·(6+8+6+6) = 6240 ns

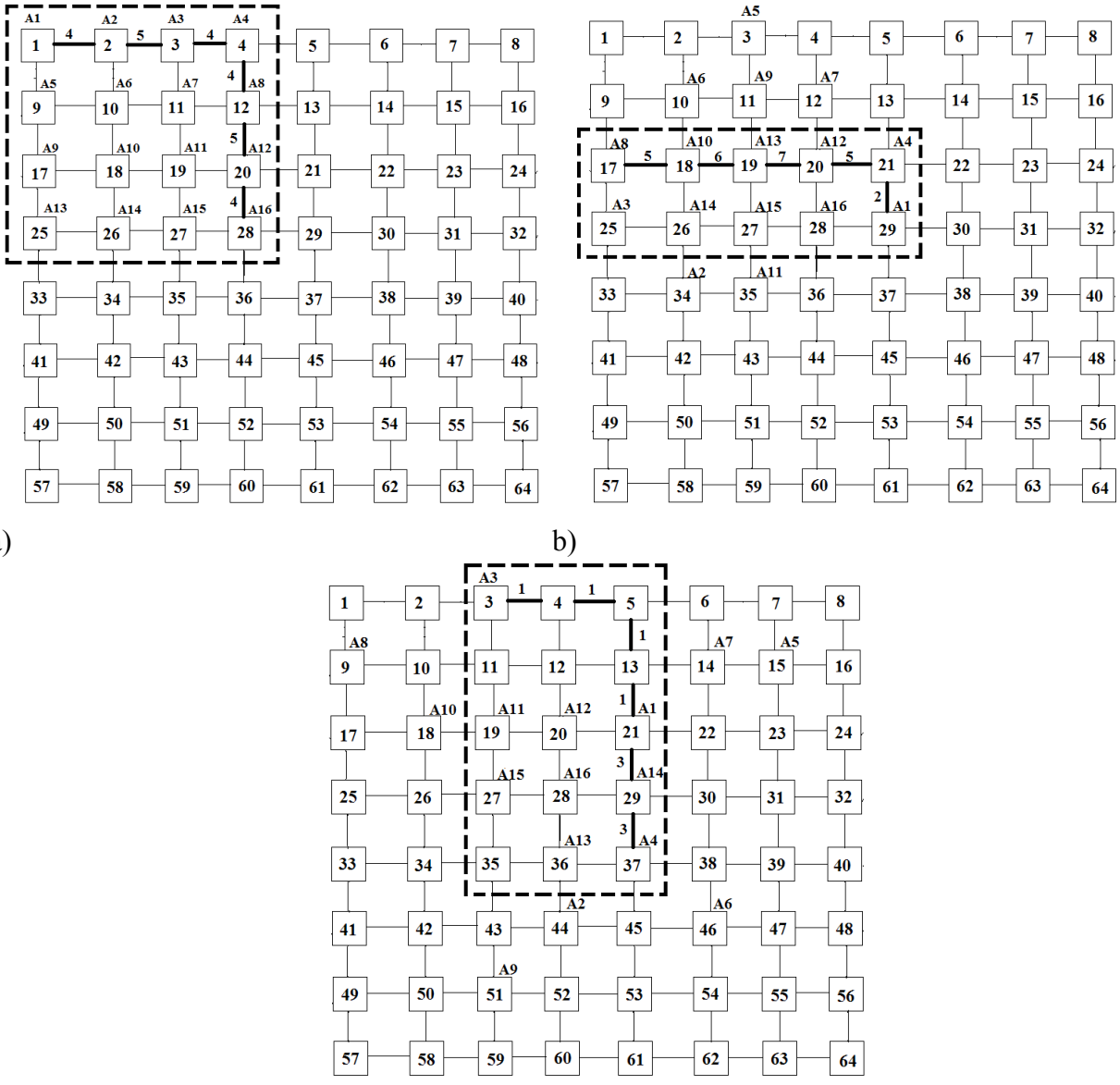

Figure 3. Initial threads-to-processors mapping for 16-body problem (a), and suboptimal threads-to-processors mappings according to minimax (b) and minimaximin (c) criteria

In Figure 3(a), the above shortest path between processors 1 and 28 is highlighted, and the number of data exchanges superimposed on each inter-processor channel are shown.

To obtain the worst case communication delay, assume the unidirectional data exchanges between processors located on different horizontal and vertical lines (Table 1) are routed along the previously specified shortest path. We then add the total delay for these data exchanges to the possible minimum as follows.

$$T_{SUM}=6240+(24\cdot2\cdot1+24\cdot3\cdot2+24\cdot4\cdot3+24\cdot5\cdot2)\cdot10 =$$
$$=6240+240\cdot(2+6+12+10)=13440 \text{ ns}$$

As shown in Figure 3(b), a suboptimal mapping is obtained using the minimax criterion to estimate mapping quality, and Figure 3(c) shows usage of the minimaximin criterion. Here, none of the obtained mappings guarantees the presence of at least one shortest path for any pair of interacting processors without overlap. Therefore, it is preferable to estimate the communication delay of the obtained suboptimal mappings using the minimaximin criterion.

According to the above, the maximum data exchange time is 8400 ns for processors 17–29 in the mapping shown in Figure 3(b) and is 4560 ns for processors 3–37 in the same shown in Figure 3(c).

As with the initial mapping, we can also estimate the minimum possible delay for the shortest paths of the processor pairs described above. In Figure 3(b), the shortest paths between processors 17 and 29 are shown inside the dashed rectangle. As shown in Figure 3(b), we assume that no processor within the outlined rectangle remains unloaded in the mapping obtained using the minimax criterion; thus, with full connectivity in the data exchange between threads, any shortest path between processors 17 and 29 can have the maximum degree of overlap. Here, consider the shortest path 17-18-19-20-21-29 whose length equals 5 hops. In this case, it is possible to select processor pairs lying on only one horizontal or one vertical line, i.e., pairs 17-18, 17-19, 17-20, 17-21, 18-19, 18-20, 18-21, 19-20, 19-21, 20-21, and 21-29. Furthermore, we can calculate the minimum possible delay for such a list of overlaps according to the lengths of the above data channels as follows.

$$T_{SUM} = (24\cdot1\cdot5+24\cdot2\cdot3+24\cdot3\cdot2+24\cdot4\cdot1+24\cdot5\cdot1)\cdot10 =$$
$$6240 \text{ ns}$$

Analogous to Figure 3(a), Figure 3(b) shows the minimum number of overlapping data exchanges on the inter-processor channels of the selected path. Here, the minimum possible delay is equal to that of the initial mapping; therefore, the mapping selected using the minimax criterion guarantees reduction of only the worst case of overlapping shortest data channels.

The shortest paths between processors 3 and 37, for which the maximum data transfer time was obtained via simulation, are within the dashed rectangle in Figure 2(b). Here, the smallest numbers of loaded processors (four) are on the shortest paths, i.e., 3-4-5-13-21-29-37, 3-4-12-13-21-29-37, and 3-11-12-13-21-29-37. Thus, the probability that these paths will be the least loaded is the highest. Among any of the abovementioned shortest routes, only processors 3, 21, 29, and 37 are loaded, with processors 21, 29, and 37 belonging to the same vertical.

We calculate the minimum possible delay between processors 3 and 37 in consideration of the lengths of overlapping channels 3-37, 21-29, 21-37, and 29-37 as follows.

$$T_{SUM} = (24\cdot6\cdot1+24\cdot1\cdot2+24\cdot2\cdot1)\cdot10 =$$
$$240\cdot(6+2+2) = 2400 \text{ ns}$$

Similar to Figures 3(a) and 3(b), Figure 3(c) shows the minimum number of overlapping data exchanges on the inter-processor channels of the selected path.

We can identify the worst case overlapping path 3-4-5-13-21-29-37 under the assumption that all unidirectional exchanges

between processors 3, 21, 29, and 37 will be on the given path. Thus, the actual overlaps along this path are 3-21, 3-29, 3-37, 21-29, 21-37, and 29-37; therefore, we obtain the following.

$$T_{SUM} = (24 \cdot 4 + 24 \cdot 5 + 24 \cdot 6 + 24 \cdot 1 \cdot 2 + 24 \cdot 2) \cdot 10 = 240 \cdot (4+5+6+2+2) = 4560 \text{ ns}$$

Since both results are less than the minimum possible delay found for processors 17-29 in the mapping selected using the minimax criterion, we conclude that it is preferable to use the minimaximin criterion when planning the mapping of parallel threads in mesh-like multiprocessors.

VIII. CONCLUSION

Based on the obtained communication delay values for the worst overlap cases, the minimum real multiprocessor performance was calculated as 0.32 GFLOP/s using the minimax criterion and 0.48 GFLOP/s using the minimaximin criterion. From these results, we conclude that the real application performance of the multiprocessor increases by 50% with the minimaximin criterion compared to the use of the minimax criterion for the target problem.

ACKNOWLEDGEMENT

The authors would like to thank the editor and anonymous reviewers for their detailed and insightful comments, which helped to significantly improve the quality of the paper.

REFERENCES

[1]. Borzov, D.B., Azzeh, I.V., Zotov, D.E., Skopin, D.M., 2014. An Approach to Achieving Increased Fault-Tolerance and Availability of Multiprocessor-Based Computer Systems. Austr. J. Basic Appl. Sci. 8, 512–522. ISSN: 1991-8178

[2]. Camara, J.M., Moreto, M., Vallejo, E., Beivide, R., Miguel-Alonso, J., Martinez, C., Navaridas, J., 2010. Twisted Torus Topologies for Enhanced Interconnection Networks. IEEE T. Parall. Distr. 21, 1765–1778. DOI: 10.1109/TPDS.2010.30

[3]. Chalasani, S., Boppana, R.V., 1997. Communication in Multicomputers with Nonconvex Faults. IEEE T. Comp. 46, 616–622. DOI: 10.1109/12.589238

[4]. Chen, D., Eisley, N., Heidelberger, P., Senger, R., Sugawara, Y., Kumar, S., Salapura, V., Satterfield, D., Steinmacher-Burow, B., Parker, J., 2012. The IBM Blue Gene/Q Interconnection Fabric. IEEE Micro 32, 32–43. DOI: 10.1109/MM.2011.96

[5]. Chmaj, G., Selvaraj, H., 2017. Interconnection Networks Efficiency in System-on-Chip Distributed Computing System: Concentrated Mesh and Fat Tree. 2017 25th International Conference on Systems Engineering (ICSEng), Las Vegas, Nevada, USA. 277–286. DOI: 10.1109/ICSEng.2017.50

[6]. DellAmico, M., Carra, D., Michiardi, P., 2016. PSBS: Practical Size-Based Scheduling. IEEE T. Comp. 65, 2199–2212. DOI: 10.1109/TC.2015.2468225

[7]. Domke, J., Hoefler, T., 2016. Scheduling-Aware Routing for Supercomputers, SC16: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Salt Lake City, UT, USA. 142–153. DOI: 10.1109/SC.2016.12

[8]. Dorigo, M., Stützle, T., 2003. The Ant Colony Optimization Metaheuristic: Algorithms, Applications and Advances. Handbook of Metaheuristics, Springer, Boston. 250–285. DOI: https://doi.org/10.1007/0-306-48056-5_9

[9]. Feil, M., Uhl, A., 2000. Multicomputer Algorithms for Wavelet Packet Image Decomposition. Parallel and Distributed Processing Symposium, International (IPDPS), Cancun, Mexico. 793. DOI: 10.1109/IPDPS.2000.846066

[10]. Friedrich, L.F., Cancian, R., de Oliveira, R.S., Corso, T.B., 2000. Performance Evaluation of Real-Time Scheduling on a Multicomputer Architecture, 2000 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS (Cat. No.00EX422) (ISPASS), Austin, TX, USA. 28–33. DOI: 10.1109/ISPASS.2000.842277

[11]. Kieu, T.C., Nguyen, K.V., Truong, N.T., Fujiwara, I., Koibuchi, M., 2016. An Interconnection Network Exploiting Trade-Off between Routing Table Size and Path Length. 2016 Fourth International Symposium on Computing and Networking (CANDAR), Hiroshima, Japan. 666–670. DOI: 10.1109/CANDAR.2016.0119

[12]. Klenke, R.H., Aylor, J.H., Han, G., 2001. Performance Modeling of Hierarchical Crossbar-Based Multicomputer Systems. IEEE T. Comp. 50, 877–890. DOI: 10.1109/12.954504

[13]. Knight, J.C., 2002. Safety Critical Systems: Challenges and Directions. Proceedings of the 24th International Conference on Software Engineering. ICSE 2002. 547–550. ISBN: 1-58113-472-X

[14]. Lau, N.D., Chien, T.Q., 2016. Solving the Traveling Salesman Problem on Multicomputer Cluster. 2016 2nd International Conference on Computational Intelligence and Networks (CINE), Bhubaneswar, India. 90–94. ISBN: 978-1-5090-0451-5

[15]. Lee, H., Jang, M., Seo, J., 2008. Petersen-Torus Networks for Multicomputer Systems. International Conference on Networked Computing and Advanced Information Management (NCM), 01, 567–571. DOI: 10.1109/NCM.2008.47

[16]. Nakano, K., Takafuji, D., Fujita, S., Matsutani, H., Fujiwara, I., Koibuchi, M., 2016. Randomly Optimized Grid Graph for Low-Latency Interconnection Networks. 2016 45th International Conference on Parallel Processing (ICPP), Philadelphia, PA, USA, 340–349. DOI: 10.1109/ICPP.2016.46

[17]. Punhani, A., Kumar, P., Nitin, A., 2017. Horizontal Fat Mesh Interconnection Network. 2017 Tenth International Conference on Contemporary Computing (IC3), Noida, India, 1–5. DOI: 10.1109/IC3.2017.8284343

[18]. Rocher-Gonzalez, J., Escudero-Sahuquillo, J., García, P.J., Quiles, F.J., 2017. On the Impact of Routing Algorithms in the Effectiveness of Queuing Schemes in High-Performance Interconnection Networks. 2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI), Santa Clara, California, USA, 65–72. DOI: 10.1109/HOTI.2017.16

[19]. Shahrabi, A., Ould-Khaoua, M., 2005. On the Performance of Routing Algorithms in Wormhole-Switched Multicomputer Networks. International Conference on Parallel and Distributed Systems (ICPADS), Fukuoka, Japan. 515–519. DOI: 10.1109/ICPADS.2005.209

[20]. Tasoulas, E., Gran, E.G., Skeie, T., Johnsen, B.D., 2016. Fast Hybrid Network Reconfiguration for Large-Scale Lossless Interconnection Networks. 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA), Cambridge, Boston, MA, USA, 101–108. DOI: 10.1109/NCA.2016.7778601

[21]. Tile processor architecture overview for the Tile-Gx series. Release 1.1, Document No. Ug130, May 2012, TILERA corporation. http://www.mellanox.com/repository/solutions/tile-scm/docs/UG130-ArchOverview-TILE-Gx.pdf [access date: 18/04/2019]

[22]. Theiss, I., Lysne, O., 2006. FRoots: A Fault Tolerant and Topology-Flexible Routing Technique. IEEE T. Parall. Distr. 17, 1136–1150. DOI: 10.1109/TPDS.2006.140

[23]. Wang G., Chen J., 2003. A New Fault-Tolerant Routing Scheme for 2-Dimensional Mesh Networks. Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, IEEE, 2003, 95–98. DOI: 10.1109/PDCAT.2003.1236266

[24]. Xiang, D., Sun, J.G., Wu, J., Thulasiraman, K., 2005. Fault-Tolerant Routing in Meshes/Tori Using Planarly Constructed Fault Blocks. Proc. Int'l Conf. Parall. Process. ICPP 2005, 577–584. DOI: 10.1109/ICPP.2005.40

[25]. Youn, H.Y., Choo, H., Yoo, S., 2000. Processor Scheduling and Allocation for 3D Torus Multicomputer Systems. IEEE T. Parall. Distr. 11, 475–484. DOI: 10.1109/71.852400

[26]. Zhu, W., Fleisch, B.D., 2000. Performance Evaluation of Soft Real-Time Scheduling for Multicomputer Cluster. Proceedings 20th IEEE International Conference on Distributed Computing Systems (ICDCS), Taipei, Taiwan, 610. DOI: 10.1109/ICDCS.2000.840977.