Snail Algorithm For Task Allocation In Mesh Networks

Bartosz Duszel
University of Nevada, Las Vegas, duszelb@unlv.nevada.edu

Follow this and additional works at: https://digitalscholarship.unlv.edu/thesesdissertations

Part of the Computer Engineering Commons, and the Electrical and Computer Engineering Commons

Repository Citation
https://digitalscholarship.unlv.edu/thesesdissertations/1983
SNAIL ALGORITHM FOR TASK ALLOCATION IN MESH NETWORKS

by

Bartosz Mikolaj Duszel

Bachelor of Science
Wroclaw University of Technology
2011

Master of Science
Wroclaw University of Technology
2012

A thesis submitted in partial fulfillment of the requirements for the

Master of Science in Electrical Engineering - Electrical Engineering

Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
December 2013
THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Bartosz Mikolaj Duszel

goingly

entitled

Snail Algorithm for Task Allocation in Mesh Networks

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering
Department of Electrical and Computer Engineering

Henry Selvaraj, Ph.D., Committee Chair
Emma Regentova, Ph.D., Committee Member
Shahram Latifi, Ph.D., Committee Member
Laxmi Gewali, Ph.D., Graduate College Representative
Kathryn Hausbeck Korgan, Ph.D., Interim Dean of the Graduate College

December 2013
Abstract

Snail Algorithm For Task Allocation In Mesh Networks

by

Bartosz Duszel

Topic of this master’s thesis is connected with task allocation algorithms and mesh networks. Author of this work has already graduated from Wroclaw, University of Technology (Poland) where during his studies he created software simulation environment for two different task allocation algorithms for mesh networks: Adaptive Scan and Frame Sliding. Those algorithms were compared by two, main parameters: simulation time and average mesh fulfillment (utilization level). All simulations were done in software environment which was developed specially for that research. This application was based on few, different types of objects: task (width, height, processing time), task queue (different number of tasks), task allocator (where different allocation strategies were implemented) and mesh structure (width, height). Whole environment was implemented using C++ language and Xcode IDE (no GUI - simulator is only a tool for this specific research, not a final product).

This work is based on three very well known task allocation algorithms: First Fit, Frame Sliding and Adaptive Scan and also one new approach (author’s own idea based on the Adaptive Scan approach) - Snail Algorithm. If new algorithm is able to scan mesh network more accurately, then tasks from the queue are allocated faster than for other algorithms (time needed for processing whole queue will be shorter). If there are more tasks on the mesh at the same time, then overall mesh utilization level (mesh fulfillment) is higher.

It was assumed that all the nodes were exactly the same and there was no delay between them so the communication was instant. This simulator is not taking into
account a lot of different parameters and delays which are however present in real life situations. For example communications delays, time needed for allocator to allocate tasks from queue on the mesh structure etc. All the experiments are based only on the execution time inside the mesh so it was easier to compare all algorithms and conclude which task arrangement is providing shorter task queue execution time and better mesh utilization level.
Acknowledgements

I would like to thank my advisor - Dr. Henry Selvaraj - who helped me a lot during my studies at the University of Nevada Las Vegas. Thanks to you I was able to come to the United States and start my second graduate program. I also want to thank a very important person in my personal life - Kamila M. - who was always there for me during my studies abroad and I was not feeling so lonely being here alone. I also want to mention my friend Piotr F. who is one of the few people who managed to stay in touch with me after moving to Las Vegas. Last but not least I am grateful to my parents for their many sided support during the years of my studies and, actually, during the whole of my life.
Dedication

To my mom and dad,

thank you for always believing in me.

I love you.
### Chapter 4  Problem Statement

4.1 Mathematical model ............................................. 23
4.2 Problem formulation ............................................. 25
4.3 Evaluation criteria ............................................. 26

### Chapter 5  Task Allocation Algorithms

5.1 Review of chosen algorithms ................................. 29
  5.1.1 Expanding Square Strategy (ESS) ......................... 29
  5.1.2 First Fit (FF) ........................................... 31
  5.1.3 Frame Sliding (FS) .................................... 33
  5.1.4 Adaptive Scan (AS) .................................. 38
  5.1.5 Snail Algorithm (new approach) ....................... 43

### Chapter 6  Simulation Environment

6.1 Simulator structure ........................................... 48
6.2 Inputs & Outputs ............................................ 51

### Chapter 7  Algorithms Comparison

7.1 Experiment 1 - mesh size .................................... 53
  7.1.1 Experiment Design ..................................... 53
  7.1.2 Results .................................................. 54
  7.1.3 Comments ............................................... 57
7.2 Experiment 2 - task queue length .......................... 59
  7.2.1 Experiment Design ..................................... 59
  7.2.2 Results .................................................. 60
  7.2.3 Comments ............................................... 63
7.3 Experiment 3 - task shapes ................................... 64
  7.3.1 Experiment Design ..................................... 65
  7.3.2 Results .................................................. 66
  7.3.3 Comments ............................................... 70
List of Tables

1  Flynn’s taxonomy ........................................ 11
2  Mesh object ............................................ 49
3  Task object ............................................. 50
4  Tasks Queue object ..................................... 50
5  AddAndRemoveFunctions ................................ 50
6  CheckFunctions ........................................ 50
7  Experiment 1 - different mesh size - input .......... 53
8  Experiment 1 - different mesh size - output ........ 54
9  Experiment 2 - different task queue length - input .. 59
10 Experiment 2 - different task queue length - output .. 60
11 Experiment 3 - different task shapes - input ........ 65
12 Experiment 3 - different task shapes - output ....... 66
13 Experiment 4 - different task sizes - input .......... 71
14 Experiment 4 - different task sizes - output ......... 72
15 Experiment 5 - different task processing time - input .. 77
16 Experiment 5 - different task processing time - output .. 78
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of transistors in CPU, source: Internet</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>IBM Deep Blue, source: Internet</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>IBM Watson playing Jeopardy!, source: Internet</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Different task parts optimization effect (sequential)</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>Amdahl’s Law</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Single Instructions Architectures</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>Multiple Instructions Architectures</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>CDC 6600, source: [1]</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>Cray-I and his creator Seymour Roger Cray, source: Internet</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>IBM BlueGene/P, source: Internet</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>Ring topology (example)</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>One complex task divided into 3 smaller subtasks.</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>$M_2(8,8)$ example</td>
<td>18</td>
</tr>
<tr>
<td>14</td>
<td>Shortest path from node A to B.</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>Mathematical model presented on block schema.</td>
<td>23</td>
</tr>
<tr>
<td>16</td>
<td>Mesh base</td>
<td>28</td>
</tr>
<tr>
<td>17</td>
<td>Expanding Square Strategy Algorithm</td>
<td>30</td>
</tr>
<tr>
<td>18</td>
<td>First Fit Algorithm</td>
<td>32</td>
</tr>
<tr>
<td>19</td>
<td>Frame Sliding Algorithm</td>
<td>35</td>
</tr>
<tr>
<td>20</td>
<td>Frame Sliding, step 2 (implemented version)</td>
<td>36</td>
</tr>
<tr>
<td>21</td>
<td>Adaptive Scan Algorithm</td>
<td>40</td>
</tr>
<tr>
<td>22</td>
<td>Adaptive Scan Algorithm (rotated task situation)</td>
<td>41</td>
</tr>
<tr>
<td>23</td>
<td>Snail Shell (algorithm name origin)</td>
<td>43</td>
</tr>
<tr>
<td>24</td>
<td>Snail Algorithm Scanning Strategy</td>
<td>45</td>
</tr>
<tr>
<td>25</td>
<td>Snail Algorithm (allocation example)</td>
<td>46</td>
</tr>
<tr>
<td>Page</td>
<td>Simulation results for different task processing time.</td>
<td>Pages</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>53</td>
<td>Simulation results for task processing time 100.</td>
<td>79</td>
</tr>
<tr>
<td>54</td>
<td>Simulation results for task processing time 150.</td>
<td>79</td>
</tr>
<tr>
<td>55</td>
<td>Simulation results for task processing time 200.</td>
<td>79</td>
</tr>
<tr>
<td>56</td>
<td>Simulation results for task processing time 250.</td>
<td>80</td>
</tr>
<tr>
<td>57</td>
<td>Simulation results for different task processing time.</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Algorithm</td>
<td>Page</td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>1</td>
<td>Expanding Square Strategy, pseudo-code</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>First Fit, pseudo-code</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>Frame Sliding, pseudo-code (modified, implemented version)</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>Adaptive Scan, pseudo-code</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>Snail Algorithm, pseudo-code</td>
<td>47</td>
</tr>
</tbody>
</table>
Chapter 1 - Introduction

1.1 Introduction

In recent decades, we have witnessed huge technological advances in many different areas of life. Not so long time ago people started using first TV sets, rotary dial telephones or general-purpose computers in their homes.

In today’s world most people carry in their pocket devices that are more like super computers of yester years. While the computational power of these devices are growing, their cost is falling. Something that was considered “state-of-art” and revolutionized people’s life 50 years ago can now be used by almost anyone and and at any place. Difference between first TV sets, telephones, computers are just examples of how fast and how big technological progress surrounds us. Another great example is Internet. People who were born 50 years ago did not even dream about such a global system that could interconnect people all over the globe. Now - it is hard to imagine world without the Internet.

Usually better quality and higher functionality goes hand in hand with higher resource requirements. For example, a faster car needs bigger (stronger) engine and more horsepower than the slower one, it also usually needs more fuel to operate. A more advanced computer is able to complete a specific task faster than a not so advanced computer thanks to more powerful processor or bigger size of memory. Higher functionality consumes more resources.

The topic of this master’s thesis is on mesh networks (that are introduced and described later in this work). In brief, it is one of many available solutions to increase system performance in those areas, where a very significant part of the network time
1.2 Motivation

During my graduate studies at Wroclaw University of Technology\(^1\), I had the opportunity to learn about mesh networks and the problem of allocating tasks on such structures. I focused in this area during my first semester of Advanced Informatics and Control program at Research Skills and Methodologies class. That time I read some already published papers connected with this topic and got familiar with some basic algorithms like:

- ESS [8],
- WSBA [16],
- WSBA2 [10].

I compared a few different task allocation solutions in different environments to see how they behaved and what the main advantages and disadvantages of those algorithms are.

At the beginning of the year (2012) I started my second graduate program. I started my master’s program at the University of Nevada Las Vegas\(^2\) (United States of America) in the Electrical and Computer Engineering\(^3\) Department. During the first semester after discussion with my supervisor - Dr. Henry Selvaraj - about my future master’s thesis, we decided that I would focus on task allocation problem.

In this work I focus on task allocation problem in mesh networks and different algorithms that can be used to make such a network more effective. This master’s

\(^1\)http://pwr.wroc.pl
\(^2\)http://unlv.edu
\(^3\)http://ece.unlv.edu
thesis should be treated as an extension to my previous work and research in this
field. Theoretical part of this work considers hardware aspects of the mesh idea but
call the results and experiments were done using software simulations.

1.3 Main goal

Main goal of this work is to explain how and why mesh networks are used all
around the world and why effective task allocation is so important in the whole com-
puting process. Many scripts (simulator) have been created to check the efficiency of
different allocation algorithms in the same test environment. Thanks to the simulator
it became possible to see how various approaches differ in final results and start more
detailed research.

Understanding how these algorithms work and knowing their advantages and dis-
advantages is crucial in deciding which algorithms are better than others under what
circumstance, environment or for what specific problem to solve..

1.4 Scope of work

This master’s thesis contains eight main sections (chapters). At the beginning
the problem area is introduced and motivations and main goals are described. Then
the background of the the presented problem is discussed, where supercomputers and
mesh idea are introduced. In this section, some information about sequential and par-
allel computing can also be found. Third chapter introduces few network topologies
that can be used to increase processing speed for task queue (mainly focusing on the
mesh idea). Fourth chapter states the problem, where mathematical model, formula-
tion of the problem and evaluation criteria are presented and explained. Then, four
chosen task allocation algorithms are explained in detail. This section is crucial to
understand the different behavior of those tasks and realize their advantages and dis-
advantages. Sixth chapter briefly describes the created simulation environment that is used to compare all task allocations algorithms described in the previous chapter. In chapter seven, all experiments are explained and results are commented. The last chapter is reserved for final conclusions and summing up all results and observations.
Chapter 2 - Problem Background

2.1 Supercomputers

During the past 60 years (since the appearance of Integrated Circuits) technology has come a long way and the efficiency of all kinds of electrical devices has improved. Moore’s Law - described by Gordon E. Moore (Intel co-founder) in his paper in 1965 states that the number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years (see Fig. 1).

It can be seen in the Intel official webpage that if the transistors were people, then in 1970s the number of transistors in an IC (Intel 4004) was equal to 2,300 that could be compared to an average music hall capacity. In 1980s (Intel 286) this number increased to 134,000 - large stadium capacity. After the year 2000 (Pentium III) 32 millions of transistors were used in a microprocessor - population of Tokyo and in 2011 (Core i7 Extreme Edition) 1.3 billion, which is approximately size of a population of China. So, basically in 31 years the number of transistors increased from 2,300 to 1,300,000 (from music hall capacity to population of China).

Increasing number of transistors on one, single chip usually does not cause proportional increase in chip size, because of compatibility with other, system elements. To increase number of transistor without changing the silicon size on the chip, dimensions of transistors must be reduced. Increasing density allows to integrate more components on a one, single die - instead of using several chips [7].

At some point our technology reached some kind of limitations. This problem is directly connected to heat that is being generated by more and more powerful chips nowadays and minimization problem (see Chapter 3.3).

Supercomputers are one of many different solutions for lagging resources or com-
putational power. Probably the easiest way of describing this concept is that "two heads are better than one". Supercomputer is based on connecting many processors into one, single network and using it to solve complex tasks. Example of such a computer is IBM Deep Blue (chess-playing computer), which on May 11, 1997 won the second six-game match against world champion Garry Kasparov.

Another example of a supercomputer is Watson (also developed by IBM) which in 2011 won the quiz show Jeopardy! against Brad Rutter (the biggest all-time money winner on Jeopardy!) and Ken Jennings (the record holder for the longest championship streak - 74 wins). Watson had access to 200 million pages that consumed four terabytes of disk storage.
Figure 2: IBM Deep Blue, *source:* Internet

Figure 3: IBM Watson playing Jeopardy! *source:* Internet

**NOTE.** parallel computing - “two heads are better than one”
2.1.1 Sequential and parallel computing

As it was mentioned before there are two, main ways to obtain higher system performance or higher computational power in general [2]. Those methods are:

- speeding up sequential computing,
- parallel computing.

First one - sequential computing - is limited by minimization and heat problems as it was already noted in 2.1. This approach is directly related to processor technology development. What is important to know is the fact that processor elements cannot be reduced indefinitely [7]. In addition, wherever higher power is consumed there is also greater necessity for heat dissipation and optimal power budget. In 2008, 16-core processor consumed about 320 watts when all cores were active (20 watts per core). Such requirements and level of consumption can exceed a single processor die’s power budget in no time [7]. Example of speeding up different, independent parts in one task can be seen in Fig. 4.

![Figure 4: Different task parts optimization effect (sequential).](image)

Second one - parallel computing - is usually faster than sequential one. It is logical that if one complex task can be divided into few simpler subtasks and calculated independently and simultaneously by many processors the whole computation time will be smaller for higher number of processors. Someone could conclude that
if such complex task needs $n$ computers (processors) to finish one, complex task in time $t$ then for $2n$ computers this time should be reduced to $\frac{t}{2}$. That is true but only for perfect situation in perfect environment. In real world for parallel computing we have to deal with some data transmission delays (even in local network) and also time reserved for task dividing process and collecting independent results from many machines.

For calculating theoretical maximum speedup of parallel computing (using multiple processors) often Amdahl’s law is being used. Basically the total speedup of a program using parallel computing is limited by the time reserved for the sequential fraction of the program. For example if 95% of the task can be parallelized then theoretical maximum performance gain using parallel computing would be 20x (no matter how many processors would be used). This dependence can be seen in Fig. 5.

### 2.1.2 Flynn’s taxonomy

One of the earliest classification systems for parallel and sequential computing was created by Michael J. Flynn\(^5\) [12]. He divided programs and computers in two groups, those which were using single set or multiple sets of instructions.

There are four, different types of machines:

- **Single Instruction Single Data (SISD)**
- **Single Instruction Multiple Data (SIMD)**
- **Multiple Instruction Single Data (MISD)**
- **Multiple Instruction Multiple Data (MIMD)**

\(^4\)Gene Amdahl (born in 1922) - American computer architect, known for his work on mainframe computers at IBM.

\(^5\)Michael J. Flynn was born in 1934 in New York City. He is an American professor emeritus at Stanford University in USA.
**SISD** - computer architecture where uniprocessor (single processor) executes a single instruction stream to operate on data which is stored in a single memory.

**SIMD** - type of computers where there is an array of processors that perform the same operation on multiple data simultaneously.

**MISD** - in this case there are different operations that are being performed on the same data.

**MIMD** - the most popular type of machines. In this architecture processors are doing different operations on different data streams.

**Figure 5: Amdahl’s Law**
Table 1: Flynn’s taxonomy

<table>
<thead>
<tr>
<th></th>
<th>Single instruction</th>
<th>Multiple instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Data</td>
<td>SISD</td>
<td>MISD</td>
</tr>
<tr>
<td>Multiple Data</td>
<td>SIMD</td>
<td>MIMD</td>
</tr>
</tbody>
</table>

(a) SISD representation

(b) SIMD representation

Figure 6: Single Instructions Architectures

(a) MISD representation

(b) MIMD representation

Figure 7: Multiple Instructions Architectures
2.1.3 History

History of supercomputers is linked to an American electrical engineering - Seymour Roger Cray\(^6\) (see Fig. 9). The first machine to be called "supercomputer" is CDC 6600 (Fig. 8) which was released in 1964 by Control Data Corporation (Seymour Cray was working there at that time). This computer was the fastest computer at that time, executing about three million instructions per second. It was the fastest computer until Seymour Cray designed CDC 7600 (five years later).

"The elegant architecture of the 6600 included one 60-bit central processor with multiple functional units coupled in parallel to ten shared-logic 12-bit peripheral I/O processors. The machine was Freon cooled.

_Selling for $6 to $10 million each, Control Data Corporation (CDC) manufactured about 100 machines [1]."

Cray left CDC in 1972 and founded Cray Research Inc. After four years he delivered Cray-1 (see Fig. 9).

Computers designed by Seymour Roger Cray were the fastest computers until United States government started ASCI (Accelerated Strategic Computing Initiative) project in 1980s. Inspiration for that program was Japan's fifth generation computer project (American government saw in the Japanese project future rival in technological dominance). Products of this project were for example Intel ASCI Red, IBM ASCI Blue, IBM ASCI White. These machines were faster than Cray's but the ASCI budget was many times larger than Cray could ever obtain.

In 1999 IBM announced a $100 million research initiative for a five-year effort to build a massively parallel computer. Name of this project was Blue Gene and first

\(^6\)Seymour Roger Cray (1925 - 1996) - American electrical engineer, "the father of supercomputing", designer responsible for many of the world's fastest computers from the 1960s to the 1980s, founder of Cray Research Inc.
Figure 8: CDC 6600, *source:* [1]

Figure 9: Cray-1 and his creator Seymour Roger Cray, *source:* Internet
version (Blue Gene/L) was introduced in 2004. It achieved first place in TOP500\textsuperscript{7} list. Blue Gene was the fastest computer in the world for 3.5 years, it was defeated by another IBM project - Roadrunner in 2008.

Currently (June 2013) the fastest computer in the world according to the TOP500 list is the **Tianhe-2** (MilkyWay-2) that has 3,120,000 cores, 1,024,000 GB of memory and consumes 17,808 kW of power. It is located in the National University of Defense Technology in China. In 2012, the number one in the list was third\textsuperscript{8} design of Blue Gene series - **Blue Gene/Q** Sequoia made by IBM (currently on third place).

![IBM BlueGene/P](source: Internet)

Figure 10: IBM BlueGene/P, *source:* Internet

\textsuperscript{7}The TOP500 project ranks and provides some additional information about most powerful known computer systems in the world.

\textsuperscript{8}Second one was Blue Gene/P, see Fig. 10.
Chapter 3 - Network Topologies

This work is focused on mesh network topology. However, it is important to know that there are many, different topologies available and each one has its own, different properties. In this section, a few selected network topologies and their network and graph theory are presented.

3.1 General definitions

Definition 3.1. **Node degree** - in graph theory, the degree of a node is the number of wires (links) that are connected to the given node.

Definition 3.2. **Path** - a set of wires that connect a sequence of vertices. Most of the time path length is given by number of hops.

Definition 3.3. **Distance** - the smallest number of wires between two nodes that have to be traversed in order to get from one processor to another. [11] [9]

Definition 3.4. **Network diameter** - maximum distance between any pair of nodes. [11] [9]

Definition 3.5. **Bisection of a network** - minimum number of wires that have to be removed in order to disconnect the network into two halves with identical (within one) number of nodes. [11] [9]

3.2 Ring

One of the simplest topologies is when each node connects to exactly two other nodes, forming a single continuous pathway. What is interesting about this topology is the fact that there is a connection between two outermost nodes that creates - a ring.
As usual there are some advantages and disadvantages with every network topology. It can be said that the advantage of ring is its very orderly network, where every node can transmit information. It is also more efficient than a bus topology under heavy network load scenario. Changing the configuration is quite easy because every node is connected to its two immediate neighbors and therefore, removing a device requires moving no more than two links (connections). It is also relatively easy to find defective node or link thanks to the point-to-point line configuration.

However, the simplest version of this topology (no redundant links) is not very reliable because when at least one of the links fails, then there would be only one more way to reach one node from another. In other words, one defective link or node can generate problems for the entire network. Communication delay is directly proportional to the number of nodes in the network because there are always only two paths to send information from one node to another. Bandwidth is shared on all links among devices.

![Figure 11: Ring topology (example)](image)

3.3 Mesh idea

When talking about computers in general, nowadays we encounter some kind of technology limitation - heat generation. Cost of designing and producing faster processors (above some level) with different solutions for the heat problem are too big and it is not profitable. That is why it is easier, faster and cheaper to use two slightly slower processors than one faster. Nowadays, we are dealing with multi-core proces-
ors or many processors in single device. The idea of mesh network is to connect several processors in one network and more effective utilization of this network in processing different tasks.

Mesh networks are used where higher computational power is required. In situations where one machine is not sufficient to solve a problem in an acceptable, the problem can be usually divided into few smaller parts (subtasks, see Fig. 12). In such case, more machines can be used to solve different subtasks. One of the possible ways of solving complex tasks is using mesh network. We present mesh network as a grid of many processors connected together [18] (see Fig. 13).

![Diagram of a mesh network](image)

Figure 12: One complex task divided into 3 smaller subtasks.

The presented network can use many different processors to solve one or many complex tasks. This method provides higher efficiency for the whole system and returns final results that are many times faster, compared to traditional methods.
Figure 13: $M_2(8, 8)$ example
3.4 Torus

In general, torus is simply an array (mesh) with wraparound links in the rows and columns [9]. It could be said that ring is a one dimensional torus. Thanks to the additional connections (compared to mesh) there are more possible paths available from one node to another (and what is important most of the time - shorter ones). Imagine how many hops are needed in $n \times n$ mesh network to send a message from bottom left corner (node A) to upper right one (node B). The path length would be at least $2n - 2$, while for the torus only 2 hops are needed, thanks to the wraparound links (see Fig. 14).
Figure 14: Shortest path from node A to B.
3.5 Network-on-Chip

While discussing multiprocessor architectures and multiprocessor networks it is important to write a few sentences on Network-on-Chip (NoC). These chips differ from standard chip-multiprocessors (CMPs) with few cores on the same die [14], [15]. In older multiprocessor systems processing elements were placed mostly on the main board and then connected by buses or network on the board (not on the chip) [3], [13].

There are a lot of benefits of adopting NoCs that usually recompensates the effort and complexity of designing and implementing such chips. The wires in the links on NoCs can be shared by multiple signals (parallelism). This feature can be called a "high level" of parallelism because all links can operate simultaneously on different data packets. Complexity of integrated systems design and architecture keeps growing and NoCs provide improved performance and scalability that are crucial in many advanced systems.

Network-on-Chip links allow to reduce the complexity of designing wires for predictable throughput, power, reliability and many more, thanks to their regular and well controlled structure. "A NoC can also provide separation between computation and communication, support modularity and IP reuse via standard interfaces, handle synchronization issues, serve as a platform for system test and increase engineering productivity."

In 2011, Altera\textsuperscript{9} published a white paper about applying the benefits of network on a chip architecture to FPGA system design, where among other things they describe the advantages of network on a chip architecture. In our opinion they explained NoC Interconnect in a very clear way and that is why a fragment of that document is cited

\textsuperscript{9}Altera Corporation is a Silicon Valley manufacturer of reconfigurable complex digital circuits like FPGAs.
"The NoC interconnect breaks the problem of communication between entities into smaller problems, such as how to transport transactions between nodes in the system, and how to encapsulate transactions into packets for transport. The NoC interconnect is different from traditional interconnects in one simple, but powerful way. Instead of treating the interconnect as a monolithic component of the system, the NoC approach treats the interconnect as a protocol stack, where different layers implement different functions of the interconnect. The power of traditional protocol stacks, such as TCP-over-IP-over-Ethernet, is that the information at each layer is encapsulated by the layer below it. The power of the Qsys NoC implementation comes from the same source, the encapsulation of information at each layer of the protocol stack." [5]
Chapter 4 - Problem Statement

4.1 Mathematical model

Before formulating the problem, mathematical model for task allocation issue is presented and explained.

The whole simulator is based on input data (task queue), which is processed by different task allocation algorithms and mesh network. After finishing the simulation the output values are saved and ready for further analysis. In other words every task allocation algorithm is working with both: Task (from Tasks Queue) and Mesh to decide where specific task should be allocated (if possible). The block schema of mathematical model can be seen in Fig. 15.

![Figure 15: Mathematical model presented on block schema.](image)

Every task $K_i$ is described by following parameters:

- $K_i$, task number ($i \in N$);
• $K_H$, task height;

• $K_W$, task width;

• $K_t$, task processing time (number of cycles needed to finish the job).

Task queue is a vector consisting of task objects, it can be described as $TQ$. Simulations run as long as this vector is still storing some tasks (when a task is completed it is being removed from the task queue). $TQ$ for $n$ tasks can be denoted as (1).

$$TQ = \sum_{i=0}^{n} K_i$$  \hspace{1cm} (1)

**NOTE.** Simulation runs as long as task queue vector is not empty. It is required to make sure that no generated (or inserted) task is wider or higher than the mesh network itself.

---

**Time of simulation** - $T$ - depends on the number of tasks $n$ in task queue $TQ$, processing time - $P_t$ - of each task and the task allocation algorithm used.

In a perfect situation, when the number of tasks in queue and dimensions of those tasks are small enough to insert all tasks on the mesh simultaneously, time of the whole simulation depends on the longest processing time and some additional cycles needed for allocating tasks in the mesh - $T_{allocation}$.

Second important simulation output is **mesh fulfillment**. This parameter provides information about temporary and average mesh utilization levels. For example, if allocation algorithm works on mesh 10x10 it has 100 nodes (processors) to choose from. If only one task from the queue is being processed on the mesh, and this
task dimensions is 6x6, then the actual mesh fulfillment is equal to 36% (because 36 processors are busy, and 64 are still free).

Basically, higher fulfillment is better and it means that whole simulation will be shorter than for lower fulfillment. If the task queue is long enough and task dimensions are different it allows task allocation algorithms to keep mesh fulfillment parameter at a relatively high level.

Current mesh fulfillment value during simulation can be denoted as (2). The average mesh fulfillment after whole simulation is obtained by dividing sum of temporary mesh fulfillment values by number of cycles of whole simulation (3).

$$f(t) = \frac{\sum_{x=1}^{X} \sum_{y=1}^{Y} q(t, p_{x,y})}{\sum_{x=1}^{X} \sum_{y=1}^{Y} p_{x,y}}$$

(2)

$p_{x,y}$ is a processor (node) in column $x$ ($x : x \in \{1, 2, \ldots, X\}$) and row $y$

$y : y \ in \{1, 2, \ldots, Y\}$

$$q(t, p_{x,y}) = \begin{cases} 0 & \text{if } p_{x,y} \text{ is free in simulation cycle } t \\ 1 & \text{if } p_{x,y} \text{ is busy in simulation cycle } t \end{cases}$$

$$F = \frac{\sum_{t=1}^{T} f(t)}{T}$$

(3)

4.2 Problem formulation

Main goal of this master’s thesis is to compare four different task allocation algorithms. Based on mathematical model, the issue was to find a task allocation, such that:

$$T = \min$$

$$F = \max$$
for given:

- queue of tasks $TQ$, where: $TQ = \sum_{i=0}^{n} K_i$,

- mesh network $M$, where: $M = \sum_{x=1}^{X} \sum_{y=1}^{Y} p_{x,y}$.

4.3 Evaluation criteria

To obtain reliable results and avoid possible variations, all the experiments scenarios - $S$ - must be repeated $n$ times. Because of limited computational power all simulations were repeated 1000 times. The average simulation result values (4) and (5) are commented after each experiment. Most of the charts however, represent the simulation results for each repetition of every simulation.

$$T_{avg} = \frac{\sum_{i=0}^{n} T_i}{n}$$ (4)

$$F_{avg} = \frac{\sum_{i=0}^{n} F_i}{n}$$ (5)

Basically algorithms are compared in two areas: time needed for completing whole task queue and mesh fulfillment. The goal of this paper is to find for which algorithm $T_{avg}$ is the lowest and for which $F_{avg}$ (mesh fulfillment) is the highest. What is more, performance of all algorithms is being checked in different scenarios and environments to see in which situations they are most effective. Such a strategy helps to decide which algorithm should be used to obtain the best performance in specific systems and for specific problems.
Chapter 5 - Task Allocation Algorithms

In this section a few task allocation algorithms are discussed. The goal of this chapter is to introduce and explain different ideas of solving task allocation problem.

Before discussing algorithms, a few definitions are introduced and explained [4].

**Definition 1** - *The base* of a sub-mesh is the processor (node) at the lower left corner of the sub-mesh.

For example, in Fig. 16 processors $<1, 0>$ and $<0, 4>$ are the bases of mesh $M_2(3, 3)$ and $M_2(2, 2)$ respectively.

**Definition 2** - *The coverage set* is a set of processors that cannot be used as base for the current task in any available sub-mesh. It is the union of all already allocated tasks. In general if $(x, y, x', y')$ is the address of an allocated sub mesh $\alpha$ and incoming task $K = (i, j)$ coverage set can be obtained by (6).

$$\left( x - i + 1, y - j + 1, x', y' \right)$$  \hspace{1cm} (6)

**Definition 3** - *A reject set* is a sub-mesh that consists of all processors which can never be used as the base of any available sub-mesh for the current task. In general, for system $M_2(w, h)$ the reject set for task $K = (i, j)$ contains two sub-meshes 7.

$$\left( w - i + 1, 0; w - 1, h - 1 \right) \text{ and } \left( 0, h - j + 1; w - 1, h - 1 \right)$$  \hspace{1cm} (7)
**Definition 4** - A *busy set* is a set of all current allocated sub-meshes in the network.

For example, the busy set in Fig. 16 consists of four node coordinates (two tasks):

\[(1, 0; 3, 2), (0, 4; 1, 5)\]

![Figure 16: Mesh base](image)
5.1 Review of chosen algorithms

5.1.1 Expanding Square Strategy (ESS)

Expanding Square Strategy algorithm was introduced in the year 2006 at the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing Conference by Seyyed-Mahmood Hosseini-Moghaddam and Mahmood Naghibzadeh [8]. The main aim of this algorithm was minimizing internal and external message-passing contention.

The authors highlight a few ESS advantages in comparison to other proposed strategies so far. First of all ESS tries to find the most compact cluster in the mesh network that results in minimizing the external message-passing contention. It also increases tasks throughput and network utilization in general. Secondly restrictions of block-based strategies can be avoided, thanks to the cluster expansion from every free processor (node) [8].

This algorithm was not implemented in the created simulator however, author wanted to mention it because of its very interesting concept and idea. ESS is completely different than all the algorithms introduced up to that point and could be used in future research.

ESS algorithm can be described in the following way [8].

- All idle (unused) processors 'build' a square around themselves.
- During each expansion all idle processors are added to their clusters.
- Expansion goes on until the number of needed processors is reached (if possible).
- If more than one cluster fulfills the requirements, then the one with minimum sum distance from all other allocated nodes in the cluster is used for task allocation.
Figure 17: Expanding Square Strategy Algorithm

Algorithm 1 Expanding Square Strategy, pseudo-code

begin
for all free processors in the system do
    cluster = {center-free-processor}
    contentionParameter = 0;
    expansion = 0;
    for each expansion ≥ 1 do
        if free processors ≤ required processors then then
            add all processors to cluster;
        else
            for each node do
                calculate allNodesDistance;
            end for
            add node with minimum allNodesDistance to cluster;
            contentionParameter = subsystemSumOfAllDistance;
        end if
    end for
    return contentionParameter;
end for
allocate job to node with minimum contentionParameter;
end

NOTE. This algorithm was not implemented in the simulator.
5.1.2 First Fit (FF)

First Fit algorithm was introduced by Yahui Zhu in his paper "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers" [17]. This algorithm is very well known thanks to its simplicity and good efficiency. Task allocation process based on two sets: reject and coverage. Algorithm searches for the first free (unallocated) processor in the mesh network. It starts horizontally from the left side to the right at the very bottom of the mesh. If no free processor is found in this row then it switches to vertical search from top to bottom at very left side of the mesh. This process repeats (with changing rows and columns) until free processor is found (or all processors will be checked). Probably the biggest disadvantage of First Fit algorithm is creation of reject and coverage sets for every task.

FF algorithm can be described in the following way.

- create coverage set for incoming task;
- create reject set for incoming task;
- starting from bottom left corner of the mesh start searching for first free node in the row;
- if there is no free processor in this row start searching first column (from the top);
- repeat for next row (and column) until free processor is found (or whole mesh is checked).

NOTE. This algorithm was implemented in the simulator.
Algorithm 2 First Fit, pseudo-code

begin
create reject set
create coverage set
for all nodes do
    if node $<x,y>$ is free AND not in created sets then
        allocate task
    else
        keep searching
    end if
end for
end
5.1.3 Frame Sliding (FS)

Frame Sliding strategy was proposed by Po-Jen Chuang and Nian-Feng Tzeng in [4]. Slightly modified version of this idea was successfully implemented in the created software.

FS algorithm can be described in 5 different steps [4]:

1. Set $i = w'$ and $j = h'$, where the current incoming task is denoted as $T = (w', h')$.

2. Generate the coverage set (based on the busy set) and reject set according to $i$ and $j$.

3. Start searching for the the lowest and leftmost available processor $< x, y >$. Check the frame base node of all candidates starting with $< x, y >$. If the base node is busy then move to the next candidate, otherwise check if frame consists of only free processors. If the whole frame is "free" go to step 4. If all available candidates have been checked and task is not allocated, go to step 5.

4. Add the current sub-mesh (frame) to the busy set and allocate task $T$ to mesh.

5. Add task $T$ to the task queue and wait until a sub-mesh is released.

However implemented algorithm is a small modification of the original solution. It is based on few steps that are listed below:

- Algorithm starts searching for the first free node in the network, starting from the bottom left corner of mesh.

- When free node is found, it is checked if whole frame (based on incoming task dimensions) consists of only free processors. First found free node is a first
candidate for the frame base\textsuperscript{10} node.

- If frame is composed of only free processor for the checked base node, incoming task is being allocated.

- If frame is not composed of only free processors, then another base node candidate is checked.

- Next base node coordinates are selected by horizontal shift (slide) of the previous position, where number of nodes shifted is equal to incoming task width.

- If last horizontal shift is made (next one would be outside mesh borders) and task is not yet allocated, base node is shifted vertically (by number of nodes equal to task height) and the search starts for very left side of mesh.

- Horizontal and vertical shifting is repeat until all candidate base nodes (according to shifting rules) are checked.

\textbf{NOTE.} Modified version of this algorithm was implemented in created simulator.

\textsuperscript{10}Bottom left corner of the frame - author reminds.
(a) Frame Sliding, step 1 (changing columns)

(b) Frame Sliding, step 2 (row change)

Figure 19: Frame Sliding Algorithm
Figure 20: Frame Sliding, step 2 (implemented version)
Algorithm 3 Frame Sliding, pseudo-code (modified, implemented version)

begin
Row = very bottom;
Column = 0; {left border}
starting from bottom, left corner of the mesh

for Row = Row - 1 do
  for Column = Column + 1 do
    if free node was founded and task dimensions can fit in this area then
      check if whole frame is free {consists of only free nodes}
      if whole frame is free and task dimensions are fine then
        insert task to the mesh;
      else
        exit this loop; {stop checking nodes one by one}
      end if
    end if
  end for
end for

for Mesh width < base node position + task width do
  shift frame base node horizontally to the right according to task width;
  if new base node is free and whole frame is also free then
    insert task to the mesh;
  end if
  if last possible base node in row was checked and frame was not completely free then
    shift frame base node vertically (according to task height) and set column number to 0;
  end if
end for
end
5.1.4 Adaptive Scan (AS)

**Adaptive Scan** algorithm was presented in 1993 at the Internal Conference on Parallel Processing by Jianxun Ding and Laxmi N. Bhuyan [6]. This algorithm is similar to Frame Sliding but it is more flexible. It basically “checks” more possible positions for incoming task in mesh than FS. It can be said that AS allows to slide frame more frequently.

The biggest difference between these algorithms is the fact that AS is shifting frame to the first, free node (FS is shifting the frame always to match incoming task width). Another difference is that when all candidates in specific row are already checked and task cannot be allocated, frame base is shifted one row up when FS is shifting frame exactly by task height size. In addition, during vertical shift original FS is not changing column number but AS is shifting frame base to the left borders of mesh so that at the end there are two shifts. However, the biggest advantage of AS algorithm is the possibility to rotate incoming task. In Fig. 22, incoming task cannot be inserted anywhere on mesh in the presented situation. After rotating the task, it can be successfully allocated. It is worth mentioning that FS algorithm would fail in this case and task would not be allocated.

AS algorithm can be described in following way.

- Check if node \(<x,y>\) is free and if frame consists of only free processors, if yes then task can be allocated. Otherwise find the nearest available free node (in the same row).

- If whole row is scanned and task cannot be allocated, shift frame base one row up and reset column number to 0 (start again from left side of the mesh).

- Repeat until task is allocated.
• If whole mesh is scanned and task is not allocated, then rotate incoming task (45 degrees) and start whole process again.

• If rotated version of incoming task is not allocated, then wait until some of the tasks are deallocated or check next task in queue.

NOTE. This algorithm was implemented in the simulator.
Figure 21: Adaptive Scan Algorithm

(a) Adaptive Scan, step 1

(b) Adaptive Scan, step 2
Figure 22: Adaptive Scan Algorithm (rotated task situation)
Algorithm 4 Adaptive Scan, pseudo-code

begin
for mesh $M = (a, b)$ and task $T = (w, h)$

STEP 1
if flag == false then
  $a' = \min(0, a - w + 1) \&\& b' = \min(0, b - h + 1)$;
else
  $a' = \min(0, a - h + 1) \&\& b' = \min(0, b - w + 1)$;
end if

STEP 2
create coverage and reject set for task $T$;

STEP 3
if node $<x, y>$ is free and is not a member of busy and / or coverage and / or reject set then
  go to step 5;
else
  d = largest x value of these sub-meshes;
  STEP 3.1
  if $x < a' - 1$ then
    $x = d + 1$ and go to step 3;
  end if
  STEP 3.2
  if $x = a' - 1$ AND $y < b' - 1$ then
    $x = 0, y = y + 1$ and go to step 3;
  end if
  STEP 3.3
  if $x = a' - 1$ AND $y = b' - 1$ AND flag == false then
    go back to step 1;
  else
    wait until a sub-mesh will be released;
  end if
end if

STEP 4
set flag = false and go back to step 1;

STEP 5
if flag == false then
  $S = (x, y, w - 1, h - 1)$;
else
  $S = (x, y, h - 1, w - 1)$;
  allocate task $T$ on mesh $M$ and add this frame to busy set;
end if
end
5.1.5 Snail Algorithm (new approach)

This chapter presents a unique Snail Algorithm for task allocation strategy. Name of the algorithm reflects the behavior of scanning the mesh. Scanning process starts at the most outer bounds of the mesh and then proceeds deeper inside the mesh structure (see Fig. 25).

Figure 23: Snail Shell (algorithm name origin)

SA (Snail Algorithm) can be described as follows:

- Algorithm starts searching for the first free node in the network, starting from the bottom left corner of the mesh.

- When a free node is found, it is checked if whole frame (based on incoming task dimensions) consists of only free processors. First free node is a the first candidate for the frame’s base node.

- If the frame is composed of only free processors for the checked base node, the task is allocated.
• If the frame is not composed of only free processors, then another candidate base node is checked.

• The process of searching next candidate base node depends on the current position of the scanning process. Snail Algorithm starts from bottom left corner of the mesh and scans the row (from left to right, see Fig. 24a).

• When the scan process reaches the processor belonging to the rejected set, it starts scanning the column (from bottom to top, see Fig. 24b).

• When it reaches the processor belonging to the rejected set, it will start scanning the row (from right to left, see Fig. 24c).

• When it reaches the left bound of the mesh it starts scanning the column (from top to bottom, see Fig. 24d).

• If the process does not end with successful task allocation, the scan process starts again but for the scanner path boundaries reduced by one (see Fig. 24e).

• The whole process is repeated until either the task allocation fails or task is allocated.

Note. This algorithm was implemented in the simulator.
(a) First Horizontal Scan

(b) First Vertical Scan

(c) Second Horizontal Scan

(d) Second Vertical Scan

(e) Whole cycle repeats for new rows and columns.

Figure 24: Snail Algorithm Scanning Strategy
Figure 25: Snail Algorithm (allocation example)
Algorithm 5  Snail Algorithm, pseudo-code

begin

deep = 0;
starting from bottom, left corner of the mesh
while deep < maxDeep do
    FIRST SCAN
    for all free nodes in a row=meshHeight-1-deep do
        if checkFrame(freeNode) == true then
            allocate the task
            break
        else
            go to next candidate
        end if
    end for

    SECOND SCAN
    for all free nodes in a column=meshWidth-taskWidth-1-deep do
        if checkFrame(freeNode) == true then
            allocate the task
            break
        else
            go to next candidate
        end if
    end for

    THIRD SCAN
    for all free nodes in a row=taskHeight-1+deep do
        if checkFrame(freeNode) == true then
            allocate the task
            break
        else
            go to next candidate
        end if
    end for

    FOURTH SCAN
    for all free nodes in a column=deep do
        if checkFrame(freeNode) == true then
            allocate the task
            break
        else
            go to next candidate
        end if
    end for
end while
end
Chapter 6 - Simulation Environment

6.1 Simulator structure

For the purpose of comparing different task allocation algorithms in the same environment for the same input data, a specific simulator has been designed and developed. It is important to know that the presented simulator itself is not treated as a final product of this master’s thesis and should not be treated so. We decided to create our own tool from the scratch to understand chosen algorithms as deeply as possible. Some more complex tools could be used to compare different task allocation algorithms, but developing our own tool allowed us to obtain better knowledge and understanding of mechanisms behind the scene. In addition, by creating a new simulator, we could choose algorithms that are actually implemented rather than being forced to use only those that are implemented in other tools. This simulator is only a helpful tool that is designed for obtaining strictly defined results for this work. As it is not meant for public domain, user friendly interface has not been developed.

The simulator consists of following classes:

- Mesh;
- Task;
- TasksQueue.

and following functionality:

- AddAndRemoveFunctions;
- CheckFunctions;
- First Fit;
• Frame Sliding;
• Adaptive Scan;
• Snail.

Each of these classes are strictly connected with objects in real world. When we are considering task allocation problems in mesh networks, we deal with tasks (Task and TasksQueue) that must be placed on the mesh (Mesh) according to task allocation algorithm (First Fit, Frame Sliding, Adaptive Scan or Snail Algorithm).

Every task in the queue contains a few different object parameters that are given in Tab. 3. Task ID is basically a unique task number starting from 0 and incremented by 1 for each new generated task in the system. Task height and width are the dimensions of the incoming job. Those information are used to tell the allocation algorithm how many processors (nodes) are needed for computing this specific, incoming task. Processing time is a parameter that informs how many cycles are needed to finish a specific job (how long this task must be processed by mesh). Logic value for rotated parameter is used by Adaptive Scan algorithm to check if the incoming task was already rotated or not. This information is important for allocating procedure to know how task should be inserted in the mesh.

Table 2: Mesh object

<table>
<thead>
<tr>
<th>Mesh object</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td>int</td>
</tr>
</tbody>
</table>

functions

- showMesh();
- clearMesh();
Table 3: Task object

<table>
<thead>
<tr>
<th>Task type</th>
<th>parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Task ID</td>
</tr>
<tr>
<td>int</td>
<td>Task Height</td>
</tr>
<tr>
<td>int</td>
<td>Task Width</td>
</tr>
<tr>
<td>int</td>
<td>2D Table</td>
</tr>
<tr>
<td>int</td>
<td>Task Processing Time</td>
</tr>
<tr>
<td>bool</td>
<td>Rotated</td>
</tr>
<tr>
<td>bool</td>
<td>Allocated</td>
</tr>
</tbody>
</table>

functions

- showTask();
- clearTask();

Table 4: Tasks Queue object

<table>
<thead>
<tr>
<th>TasksQueue type</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Simulation Time</td>
</tr>
<tr>
<td>int</td>
<td>Average Mesh Fullfilment</td>
</tr>
<tr>
<td>vector&lt;Task&gt;</td>
<td>queue</td>
</tr>
</tbody>
</table>

functions

- showTasksQueue();

Table 5: AddAndRemoveFunctions

AddAndRemoveFunctions functions

- insertTask();
- insertRotatedTask();
- releaseFinishedTasks;
- releaseFinishedTasksWithNumberOfTasks();

Table 6: CheckFunctions

CheckFunctions functions

- checkIfTaskCanBeAlreadyReleased();
- checkFrame();
- checkRotatedFrame;
- updateMeshFulfillment();
6.2 Inputs & Outputs

Created simulation environment allows us to change many different simulation inputs like:

- number of simulations,
- number of tasks in queue,
- tasks dimensions,
- mesh dimensions,
- tasks processing time,
- task allocation algorithm.

Average mesh fulfillment and total simulation time are calculated as outputs. These results are saved in two files for each algorithm: *.txt and *.csv. Information about experiment design (simulation inputs and global parameters) is in the text file, *.csv file contains two columns: simulation time and average mesh fulfillment for each simulation. These files are used for creating charts that are later in this work. Simulator block schema representation can be seen in Fig. 26.

Figure 26: Simulator presented on block schema.
Chapter 7 - Algorithms Comparison

In this chapter, all obtained simulation results are presented and commented. For this master’s thesis five different experiments have been performed to check which task allocation algorithm is better in what circumstances. Results from the first experiment are based on changing mesh dimensions (rest of the simulation parameters are constant). Second experiment is checking the influence of different number of tasks in queue (queue length) on algorithm’s efficiency. Third scenario is about different task shapes. Main purpose of testing task shapes is to the Adaptive Scan rotation feature and its effects. The simulator works with different task sizes (but with keeping square shape) in experiment number four. The last experiment is about different task processing time.

This chapter is constructed in the following way: At the beginning of each experiment, simulation inputs and experiment design are presented. Results of all experiments are presented using tables (parameter values) and charts (for each experiment scenario). At the end of each experiment section, results are summarized and commented.
7.1 Experiment 1 - mesh size

7.1.1 Experiment Design

First experiment checks the influence of different mesh network dimensions - $M_2(n,n)$ - on total simulation time - $T$ - and average mesh fulfillment - $F$ - for all four algorithms. It is logical that for the static task queue length the simulation time should be smaller for bigger mesh networks. This is due to the fact that as long as the task parameters in the queue are constant, for bigger mesh more tasks can be allocated and processed simultaneously.

<table>
<thead>
<tr>
<th>SIMULATION INPUT</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesh dimensions</td>
<td>different</td>
</tr>
<tr>
<td>number of tasks in queue</td>
<td>1000</td>
</tr>
<tr>
<td>number of simulations</td>
<td>1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>parameter</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>task width ($K_W$)</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>task height ($K_H$)</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>task processing time ($K_t$)</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 7: Experiment 1 - different mesh size - input
7.1.2 Results

In this subsection, simulation results are presented. All observations and conclusions are in subsection 7.1.3.

Table 8: Experiment 1 - different mesh size - output

<table>
<thead>
<tr>
<th>Mesh Dimensions</th>
<th>SIMULATION OUTPUT</th>
<th>FF</th>
<th>FS</th>
<th>AS</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>20x20</td>
<td>average simulation time</td>
<td>18706</td>
<td>14975</td>
<td>13365</td>
<td>9429</td>
</tr>
<tr>
<td></td>
<td>average mesh fulfillment</td>
<td>28%</td>
<td>34%</td>
<td>35%</td>
<td>45%</td>
</tr>
<tr>
<td>30x30</td>
<td>average simulation time</td>
<td>15208</td>
<td>10055</td>
<td>11188</td>
<td>6853</td>
</tr>
<tr>
<td></td>
<td>average mesh fulfillment</td>
<td>16%</td>
<td>22%</td>
<td>20%</td>
<td>27%</td>
</tr>
<tr>
<td>40x40</td>
<td>average simulation time</td>
<td>13672</td>
<td>9786</td>
<td>9336</td>
<td>6228</td>
</tr>
<tr>
<td></td>
<td>average mesh fulfillment</td>
<td>10%</td>
<td>13%</td>
<td>13%</td>
<td>17%</td>
</tr>
<tr>
<td>50x50</td>
<td>average simulation time</td>
<td>13336</td>
<td>9688</td>
<td>8009</td>
<td>5865</td>
</tr>
<tr>
<td></td>
<td>average mesh fulfillment</td>
<td>7%</td>
<td>9%</td>
<td>10%</td>
<td>12%</td>
</tr>
<tr>
<td>60x60</td>
<td>average simulation time</td>
<td>13320</td>
<td>9612</td>
<td>7051</td>
<td>5612</td>
</tr>
<tr>
<td></td>
<td>average mesh fulfillment</td>
<td>5%</td>
<td>7%</td>
<td>8%</td>
<td>9%</td>
</tr>
</tbody>
</table>

(a) Average simulation time for 20x20 mesh network.  
(b) Average mesh fulfillment level for 20x20 mesh network.

Figure 27: Simulation results for 20x20 mesh network.
(a) Average simulation time for 30x30 mesh network.
(b) Average mesh fulfillment level for 30x30 mesh network.

Figure 28: Simulation results for 30x30 mesh network.

(a) Average simulation time for 40x40 mesh network.
(b) Average mesh fulfillment level for 40x40 mesh network.

Figure 29: Simulation results for 40x40 mesh network.

(a) Average simulation time for 50x50 mesh network.
(b) Average mesh fulfillment level for 50x50 mesh network.

Figure 30: Simulation results for 50x50 mesh network.
Figure 31: Simulation results for 60x60 mesh network.

(a) Average simulation time for 60x60 mesh network.
(b) Average mesh fulfillment level for 60x60 mesh network.

Figure 32: Simulation results for different mesh networks.

(a) Average simulation time for different mesh networks.
(b) Average mesh fulfillment level for different mesh networks.
7.1.3 Comments

As expected, task queue is processed faster for bigger mesh networks for all algorithms. In Tab. 8 and Fig. 32a it can be seen how big the influence of changing the network dimensions is for different algorithms. For $20 \times 20$ network, task queue is processed after 18706 simulation cycles for First Fit, after 14975 for Frame Sliding, 13365 for Adaptive Scan and finally only 9426 simulation cycles were needed for Snail Algorithm. So, in this case Snail Algorithm is almost two times faster than First Fit. For four times bigger mesh - $40 \times 40$ - average simulation time decreases to 13672 for First Fit, 9786 for Frame Sliding, 9336 for Adaptive Scan and 6228 for Snail Algorithm. However there is no really big difference for increasing mesh network any further. In Fig. 30 and 31 first two algorithms (FF and FS) provide almost the same results for both scenarios and the other two algorithms (AS, SA) provide only slightly improved results. Such a behavior shows that in this specific case (maximum task dimensions are equal to $10 \times 10$ and tasks processing time is relatively small) the most optimal solution for all algorithms is to use mesh dimensions about four times larger than maximum task dimensions in given queue.

The highest mesh fulfillment level is obtained for the first simulation scenario, where mesh network dimensions are equal to $20 \times 20$. Maximum possible task dimensions are set to $10 \times 10$ so that a relatively worse scenario such a mesh can be still fully filled with four tasks. For the First Fit, mesh utilization level was around 28%, for Frame Sliding 34%, Adaptive Scan 35% and the highest for Snail Algorithm - 45%. Those values constantly decrease for bigger mesh networks and for the mesh dimensions equal to $60 \times 60$ they are around 5% for FF, 7% for FS, 8% for AS and 9% for SA. Such results are strongly (but not only) influenced by task processing time. For smaller task processing time it is possible that after allocating one task there is not ‘enough time’ to allocate all other possible tasks because during this process
the first task needs to be already deallocated. If the maximum task processing time
is bigger, then all the algorithms would have more time to allocate as many tasks
as possible on the mesh structure. This problem is however, tested in experiment
number 5 (section 7.5).
7.2 Experiment 2 - task queue length

Second experiment is on task queue length (number of tasks in generated queue). Similar to previous experiment, the number of simulations is equal to 1000 but mesh dimensions - $M_2(n,n)$ - are this time fixed and equal to $20 \times 20$. Number of tasks in queue - $TQ$ - is changed for different scenarios (from 500 to 2500).

It can be predicted that for longer queue - $TQ$ - total simulation time - $T$ - should increase. Average mesh fulfillment - $F$ - should be similar for all scenarios because the only thing that is being changed is the number of tasks in queue. Tasks are generated in the same way as in experiment 1 (dimensions $K_W$, $K_H$ and task processing time).

7.2.1 Experiment Design

In this subsection, simulation results are presented. All observations and conclusions are in subsection 7.2.3.

Table 9: Experiment 2 - different task queue length - input

<table>
<thead>
<tr>
<th>SIMULATION INPUT</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesh dimensions</td>
<td>20x20</td>
</tr>
<tr>
<td>number of tasks in queue</td>
<td>different</td>
</tr>
<tr>
<td>number of simulations</td>
<td>1000</td>
</tr>
<tr>
<td>task width ($K_W$)</td>
<td>1</td>
</tr>
<tr>
<td>task height ($K_H$)</td>
<td>1</td>
</tr>
<tr>
<td>task processing time ($K_t$)</td>
<td>1</td>
</tr>
</tbody>
</table>
### 7.2.2 Results

Table 10: Experiment 2 - different task queue length - output

<table>
<thead>
<tr>
<th>SIMULATION OUTPUT</th>
<th>FF</th>
<th>FS</th>
<th>AS</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task Queue Length</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>9372</td>
<td>7538</td>
<td>6700</td>
<td>4729</td>
</tr>
<tr>
<td>average simulation time</td>
<td>9372</td>
<td>7538</td>
<td>6700</td>
<td>4729</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>28%</td>
<td>34%</td>
<td>35%</td>
<td>45%</td>
</tr>
<tr>
<td>1000</td>
<td>18715</td>
<td>14985</td>
<td>13372</td>
<td>9431</td>
</tr>
<tr>
<td>average simulation time</td>
<td>18715</td>
<td>14985</td>
<td>13372</td>
<td>9431</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>28%</td>
<td>34%</td>
<td>35%</td>
<td>45%</td>
</tr>
<tr>
<td>1500</td>
<td>28046</td>
<td>22431</td>
<td>20059</td>
<td>14115</td>
</tr>
<tr>
<td>average simulation time</td>
<td>28046</td>
<td>22431</td>
<td>20059</td>
<td>14115</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>28%</td>
<td>34%</td>
<td>35%</td>
<td>45%</td>
</tr>
<tr>
<td>2000</td>
<td>37395</td>
<td>29940</td>
<td>26724</td>
<td>18818</td>
</tr>
<tr>
<td>average simulation time</td>
<td>37395</td>
<td>29940</td>
<td>26724</td>
<td>18818</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>28%</td>
<td>34%</td>
<td>35%</td>
<td>45%</td>
</tr>
<tr>
<td>2500</td>
<td>46711</td>
<td>37388</td>
<td>33413</td>
<td>23507</td>
</tr>
<tr>
<td>average simulation time</td>
<td>46711</td>
<td>37388</td>
<td>33413</td>
<td>23507</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>28%</td>
<td>34%</td>
<td>35%</td>
<td>45%</td>
</tr>
</tbody>
</table>

(a) Average simulation time for 500 tasks in the queue.

(b) Average mesh fulfillment level for 500 tasks in the queue.

Figure 33: Simulation results for 500 tasks in the queue.
(a) Average simulation time for 1000 tasks in the queue.
(b) Average mesh fulfillment level for 1000 tasks in the queue.

Figure 34: Simulation results for 1000 tasks in the queue.

(a) Average simulation time for 1500 tasks in the queue.
(b) Average mesh fulfillment level for 1500 tasks in the queue.

Figure 35: Simulation results for 1500 tasks in the queue.

(a) Average simulation time for 2000 tasks in the queue.
(b) Average mesh fulfillment level for 2000 tasks in the queue.

Figure 36: Simulation results for 2000 tasks in the queue.
(a) Average simulation time for 2500 tasks in the queue.  
(b) Average mesh fulfillment level for 2500 tasks in the queue.

Figure 37: Simulation results for 2500 tasks in the queue.

(a) Average simulation time for different tasks in the queues.  
(b) Average mesh fulfillment level for different number of tasks in the queues.

Figure 38: Simulation results for different number of tasks in the queues.
7.2.3 Comments

Obtained results have been correctly predicted in the experiment description. It is logical that for bigger number of tasks (longer queue) and fixed mesh dimensions total simulation time - $T$ - should increase. In Fig. 38a the relationship between task queue length and simulation time (and average mesh fulfillment level) can be clearly seen. For all queues Snail Algorithm is able to finish simulation before all other algorithms. For shortest queue (500 tasks) the average simulation time of 1000 simulations is equal to 4729 for Snail Algorithm. Adaptive Scan is slower and needs 6700 simulation cycles to finish the simulation. Frame Sliding - 7538 - is still better than First Fit that required 9372 simulation cycles to process the queue.

For all scenarios, average mesh fulfillment - $F$ - is fixed for all algorithms (as expected). For First Fit, it is equal to 28%, for Frame Sliding 34%, 35% for Adaptive Scan and finally 45% for Snail Algorithm. As in experiment 1, Snail Algorithm provides the best results (shortest simulation time and highest mesh fulfillment level).
7.3 Experiment 3 - task shapes

Different task shapes are tested in this experiment. The main goal of this experiment is to show when and how the task rotating feature of Adaptive Scan algorithm can be used. It is important to understand the difference between task shapes and task sizes (dimensions) that are tested in experiment 4 (next subsection). Up to this point task dimension values have been generated using pseudo-random functions available in C++ language. Those values are randomly drawn from 1 to maximum 10 processors for both: width ($K_W$) and height ($K_H$). However, for the next two experiments those parameters are fixed in order to force testing of different task shapes and their influence on task allocation algorithms. Adaptive Scan is the only algorithm (implemented in the simulator) that can rotate task if it cannot be allocated in the original version.

Results of this experiment should confirm that Adaptive Scan algorithm can provide higher average mesh fulfillment parameter - $F$ - and shorter simulation time - $T$ - thanks to the ability of rotating tasks from the queue. There are different scenarios when rotated task can be allocated on the mesh but original versions of the tasks cannot (and when other algorithms fail and have to wait until some other tasks are released that are still under processing in the mesh).
7.3.1 Experiment Design

In this subsection, simulation results are presented. All observations and conclusions are in subsection 7.3.3.

Table 11: Experiment 3 - different task shapes - input

<table>
<thead>
<tr>
<th>SIMULATION INPUT</th>
<th>parameter</th>
<th>value</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mesh dimensions</td>
<td>20x20</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>number of tasks in queue</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>number of simulations</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>task width ((K_w))</td>
<td>different</td>
<td>different</td>
<td></td>
</tr>
<tr>
<td></td>
<td>task height ((K_H))</td>
<td>different</td>
<td>different</td>
<td></td>
</tr>
<tr>
<td></td>
<td>task processing time ((K_t))</td>
<td>1</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
7.3.2 Results

Table 12: Experiment 3 - different task shapes - output

<table>
<thead>
<tr>
<th>Task Dimensions</th>
<th>FF</th>
<th>FS</th>
<th>AS</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>10x10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>35012</td>
<td>41929</td>
<td><strong>13188</strong></td>
<td>15044</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>43%</td>
<td>53%</td>
<td><strong>75%</strong></td>
<td>67%</td>
</tr>
<tr>
<td>11x1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>16745</td>
<td>18859</td>
<td><strong>2148</strong></td>
<td>3009</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>15%</td>
<td>16%</td>
<td><strong>69%</strong></td>
<td>50%</td>
</tr>
<tr>
<td>11x3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>28845</td>
<td>34661</td>
<td><strong>5932</strong></td>
<td>9887</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>22%</td>
<td>24%</td>
<td><strong>66%</strong></td>
<td>48%</td>
</tr>
<tr>
<td>11x5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>36384</td>
<td>48947</td>
<td><strong>10573</strong></td>
<td>17040</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>27%</td>
<td>32%</td>
<td><strong>59%</strong></td>
<td>47%</td>
</tr>
<tr>
<td>11x7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>41636</td>
<td>78841</td>
<td><strong>17552</strong></td>
<td>28763</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>32%</td>
<td>17%</td>
<td><strong>44%</strong></td>
<td>41%</td>
</tr>
<tr>
<td>11x9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>47877</td>
<td>73596</td>
<td><strong>17558</strong></td>
<td>31569</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>35%</td>
<td>21%</td>
<td><strong>52%</strong></td>
<td>47%</td>
</tr>
</tbody>
</table>

(a) Average simulation time for tasks 10 × 10.  (b) Average mesh fulfillment level for tasks 10 × 10.

Figure 39: Simulation results for tasks 10 × 10.
(a) Average simulation time for tasks $11 \times 1$.  (b) Average mesh fulfillment level for tasks $11 \times 1$.

Figure 40: Simulation results for tasks $11 \times 1$.

(a) Average simulation time for tasks $11 \times 3$.  (b) Average mesh fulfillment level for tasks $11 \times 3$.

Figure 41: Simulation results for tasks $11 \times 3$.

(a) Average simulation time for tasks $11 \times 5$.  (b) Average mesh fulfillment level for tasks $11 \times 5$.

Figure 42: Simulation results for tasks $11 \times 5$. 
(a) Average simulation time for tasks $11 \times 7$.  
(b) Average mesh fulfillment level for tasks $11 \times 7$.

Figure 43: Simulation results for tasks $11 \times 7$.

---

(a) Average simulation time for tasks $11 \times 9$.  
(b) Average mesh fulfillment level for tasks $11 \times 9$.

Figure 44: Simulation results for tasks $11 \times 9$.
Figure 45: Simulation results for different task shapes.

(a) Average simulation time for different task shapes.

(b) Average mesh fulfillment level for different task shapes.
7.3.3 Comments

In Tab. 12, we see that mesh fulfillment is the highest for tasks with regular, square shape. This experiment was run on mesh $20 \times 20$ and therefore, it was possible to fill the whole mesh with four $10 \times 10$ tasks and keep mesh utilization at high level. With four tasks allocated, mesh fulfillment is equal to 100%, however it is not possible to obtain exactly 100%. However, it is not possible to obtain exactly 100% at the end of the whole simulations because there are still some simulation cycles needed for deallocation process, mesh scanning (for incoming task) etc.

Adaptive Scan in this case is the fastest algorithm, Snail Algorithm provides better results than First Fit and Frame Sliding (both: simulation time and average mesh fulfillment). All algorithms are able to allocate four $10 \times 10$ tasks on the mesh but because of different strategies it takes them different amount of time to scan mesh and find correct position for root nodes. This is why both: mesh fulfillment level and simulation time are different for those algorithms. If one algorithm (like First Fit) requires more "time" for scanning the mesh (because it checks every node in the mesh that is slower than for example, sliding the task frame) then free processors are unused for longer periods of time that generates lower mesh fulfillment level.

The fastest simulations were reached by tasks $11 \times 1$ and $11 \times 3$. Width of the tasks was fixed but for increasing height ($11 \times 5$, $11 \times 7$, $11 \times 9$) simulation time was also increasing. It is logical that if tasks are getting bigger and mesh dimensions are constant, then less number of tasks can be allocated on the mesh simultaneously so more time is needed to finish the whole task queue. Adaptive Scan was able to keep more tasks on the mesh thanks to its rotating ability and this is why this algorithm provides the best results for such queue.
7.4 Experiment 4 - task sizes

Previous experiment showed that Adaptive Scan’s rotating ability can be used to provide better results than algorithms without such features. In this experiment shapes of the task - in contrast to experiment 3 - are always square ($K_W = K_H$). Mesh dimensions - $M_2(n, n)$ - are still $20 \times 20$, task queue length is equal to 1000 and all scenarios are run 1000 times. For tasks with square shapes Adaptive Scan’s rotating ability does not have any use because rotation does not change anything in those cases.

7.4.1 Experiment Design

In this subsection, simulation results are presented. All observations and conclusions are in subsection 7.4.3.

Table 13: Experiment 4 - different task sizes - input

<table>
<thead>
<tr>
<th>SIMULATION INPUT</th>
<th>value</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mesh dimensions</td>
<td>20x20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of tasks in queue</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of simulations</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>task dimension (square)</td>
<td>different</td>
<td></td>
<td></td>
</tr>
<tr>
<td>task processing time ($K_t$)</td>
<td>1</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Table 14: Experiment 4 - different task sizes - output

<table>
<thead>
<tr>
<th>SIMULATION OUTPUT</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>FF</td>
<td>FS</td>
<td>AS</td>
<td>SA</td>
</tr>
<tr>
<td>Task Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2x2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>8399</td>
<td>8667</td>
<td>2085</td>
<td>2324</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>12%</td>
<td>12%</td>
<td>26%</td>
<td>23%</td>
</tr>
<tr>
<td>4x4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>13293</td>
<td>16509</td>
<td>3132</td>
<td>4136</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>21%</td>
<td>20%</td>
<td>57%</td>
<td>51%</td>
</tr>
<tr>
<td>5x5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>16295</td>
<td>21085</td>
<td>4578</td>
<td>5630</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>26%</td>
<td>25%</td>
<td>59%</td>
<td>58%</td>
</tr>
<tr>
<td>6x6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>20429</td>
<td>27349</td>
<td>7713</td>
<td>8574</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>30%</td>
<td>29%</td>
<td>57%</td>
<td>57%</td>
</tr>
<tr>
<td>8x8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>28592</td>
<td>39351</td>
<td>13179</td>
<td>16735</td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>36%</td>
<td>38%</td>
<td>54%</td>
<td>52%</td>
</tr>
</tbody>
</table>

(a) Average simulation time for tasks $2 \times 2$.  
(b) Average mesh fulfillment level for tasks $2 \times 2$.

Figure 46: Simulation results for tasks $2 \times 2$. 
Figure 47: Simulation results for tasks $4 \times 4$.

Figure 48: Simulation results for tasks $5 \times 5$.

Figure 49: Simulation results for tasks $6 \times 6$. 

(a) Average simulation time for tasks $4 \times 4$.

(b) Average mesh fulfillment level for tasks $4 \times 4$.

(a) Average simulation time for tasks $5 \times 5$.

(b) Average mesh fulfillment level for tasks $5 \times 5$.

(a) Average simulation time for tasks $6 \times 6$.

(b) Average mesh fulfillment level for tasks $6 \times 6$. 

73
(a) Average simulation time for tasks $8 \times 8$.  
(b) Average mesh fulfillment level for tasks $8 \times 8$.

Figure 50: Simulation results for tasks $8 \times 8$.

(a) Average simulation time for different task sizes.

(b) Average mesh fulfillment level for different task sizes.

Figure 51: Simulation results for different task sizes.
7.4.3 Comments

Results from this experiment are very interesting as they show that for faster algorithms, mesh fulfillment parameter increases for tasks dimensions $2 \times 2$, $4 \times 4$ and $5 \times 5$. However, for the next two scenarios it starts to fall (Tab. 14). For all simulations mesh dimensions are equal to $20 \times 20$ so that task with dimensions $2 \times 2$, $5 \times 5$ and $5 \times 5$ can perfectly fill the whole mesh. Of course, this holds as long as the processing time is high enough. If required processing time is too short then average mesh fulfillment level is lower than it could be. This happens in situations when some previous tasks are ready for deallocation before all possible tasks from the queue are allocated. This is why there is a huge difference between first two scenarios in this experiment - for example it is possible to allocate two times more $2 \times 2$ tasks than $5 \times 5$ on the $20 \times 20$ mesh.

For the other two scenarios ($6 \times 6$, $8 \times 8$) simulation time increases for all algorithms (what is intuitive) but mesh fulfillment for Adaptive Scan and Snail Algorithm decreases. Such a behavior is related to task dimensions that cannot fully cover the mesh network anymore (there will be always some free, unused nodes). However, the average mesh fulfillment level for the first two algorithms - First Fit and Frame Sliding - still increases. This is because those algorithms are many times slower than the other two. When Adaptive Scan and Snail Algorithm are already done with allocating process for n tasks (no more tasks can be allocated for current mesh state), First Fit and Frame Sliding are still in the network scanning process and keeps allocating the tasks from the queue. The "filling" process is many times slower and this keeps the - F - at relatively low level. However, for bigger tasks, every successful allocation covers bigger number of nodes that keeps the mesh utilization level higher than for smaller tasks.
7.5 **Experiment 5 - tasks processing time**

From the analysis and discussion of results from all the previous experiments, it is noted that average mesh fulfillment parameter - $F$ - should be bigger for queues with tasks requiring longer processing time to be finished and deallocated from the mesh. With relatively small task processing time - $K_t$ - task allocation algorithm does not have enough "time" to insert all possible tasks on the mesh. While all those tasks are picked up from the queue and the allocation algorithm scanned the whole mesh for free space, it is possible that previously allocated tasks could become ready for the deallocation process. Thanks to longer task processing time for each task from the queue, it is possible for the allocation algorithm to scan and insert more tasks on the mesh. This results in higher average mesh fulfillment.

It is also logical that for longer task processing time for each task from the queue, the whole simulation time becomes higher. Queue length (number of tasks) is fixed and equal to 1000, mesh dimensions and maximum task width and height are also fixed.
### 7.5.1 Experiment Design

In this subsection, simulation results are presented. All observations and conclusions are in subsection 7.5.3.

Table 15: Experiment 5 - different task processing time - input

<table>
<thead>
<tr>
<th>SIMULATION INPUT</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesh dimensions</td>
<td>20x20</td>
</tr>
<tr>
<td>number of tasks in queue</td>
<td>1000</td>
</tr>
<tr>
<td>number of simulations</td>
<td>1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>parameter</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>task width ($K_W$)</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>task height ($K_H$)</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>task processing time ($K_t$)</td>
<td>1</td>
<td>different</td>
</tr>
</tbody>
</table>
7.5.2 Results

Table 16: Experiment 5 - different task processing time - output

<table>
<thead>
<tr>
<th>SIMULATION OUTPUT</th>
<th>parameter</th>
<th>FF</th>
<th>FS</th>
<th>AS</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task Processing Time</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>11797</td>
<td>10427</td>
<td>8810</td>
<td>5938</td>
<td></td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>24%</td>
<td>29%</td>
<td>29%</td>
<td>37%</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>18715</td>
<td>15005</td>
<td>13361</td>
<td>9419</td>
<td></td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>28%</td>
<td>34%</td>
<td>35%</td>
<td>45%</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>24368</td>
<td>19814</td>
<td>17003</td>
<td>12971</td>
<td></td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>31%</td>
<td>36%</td>
<td>40%</td>
<td>49%</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>29308</td>
<td>24827</td>
<td>20272</td>
<td>16472</td>
<td></td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>33%</td>
<td>38%</td>
<td>44%</td>
<td>51%</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average simulation time</td>
<td>33720</td>
<td>29938</td>
<td>23440</td>
<td>19908</td>
<td></td>
</tr>
<tr>
<td>average mesh fulfillment</td>
<td>35%</td>
<td>38%</td>
<td>47%</td>
<td>53%</td>
<td></td>
</tr>
</tbody>
</table>

(a) Average simulation time for task processing time 50.
(b) Average mesh fulfillment level for task processing time 50.

Figure 52: Simulation results for task processing time 50.
(a) Average simulation time for task processing time 100.

(b) Average mesh fulfillment level for task processing time 100.

Figure 53: Simulation results for task processing time 100.

(a) Average simulation time for task processing time 150.

(b) Average mesh fulfillment level for task processing time 150.

Figure 54: Simulation results for task processing time 150.

(a) Average simulation time for task processing time 200.

(b) Average mesh fulfillment level for task processing time 200.

Figure 55: Simulation results for task processing time 200.
(a) Average simulation time for task processing time 250.  
(b) Average mesh fulfillment level for task processing time 250.

Figure 56: Simulation results for task processing time 250.

(a) Average simulation time for different task processing time.

(b) Average mesh fulfillment level for different task processing time.

Figure 57: Simulation results for different task processing time.
7.5.3 Comments

The last experiment was based on different task processing time. As with task queue length, it can be predicted that for tasks that require more processing time, the whole simulation time - $T$ - should be higher because each of the tasks in the queue needs to stay on the mesh for longer period of time. Results of all the experiments show that this is true and for longer processing time the average simulation time increases.

Average mesh fulfillment level - $F$ - also increases with task processing time. This behavior has been already examined in this work and can be explained as "time" that the algorithm requires for allocating incoming task from the queue. If the task processing time is too short, then allocation algorithm does not have enough "time" to scan the whole mesh and allocate all possible tasks for the current state of the mesh. If tasks need to be processed for long enough, then the algorithm can scan the whole mesh and - if possible - allocate incoming tasks without deallocating previous tasks. It also helps to keep mesh utilization level on higher level.

The Snail Algorithm provided the best results in this experiment. It is able to process the whole queue in the shortest time and keeps the highest mesh average utilization level. For a task processing time equal to 50 simulation cycles, the average simulation time is around 6000 with mesh utilization level at 37%. For the same task queue, Adaptive Scan is 50% slower (needs almost 9000 simulation cycles) and keeps the mesh at 25% fulfillment level.
Chapter 8 - Conclusions

Main goal of this thesis is to compare three very well known task allocation algorithms: First Fit, Frame Sliding and Adaptive Scan with a new approach: Snail Algorithm. The selected algorithms have been presented and explained in detail in section 5. For the research purpose, more than 25,000 simulations were completed to provide average results for section 7. In each simulation, efficiency of all four algorithms was tracked using simulation time \((T)\) and average mesh fulfillment \((F)\) values. Simulations were divided into five experiments each consisted of five scenarios. Experiments were designed in such a way to generate relatively large range of possible input data (task queues) and 'problematic' situations for task allocation algorithms.

The first experiment demonstrated mesh dimensions for which a task queue can be processed in the most efficient way. As it was expected, for larger network the total simulation time decreases. There is more space for allocating the tasks so more tasks can be allocated and processed simultaneously. However, for too big networks, the average mesh fulfillment level can be relatively low because it will be impossible for the task allocation algorithms to scan the whole mesh without deallocating previous tasks.

In the second experiment, the length of the task queue is changed for different simulation scenarios. It was anticipated that for longer queue the simulation time would increase. There is more "work" to do and therefore, more time is needed to process all the tasks. However, the average mesh utilization level is fixed for all scenarios because parameters of the tasks itself’s are not changed.

Experiment number three is focused on different task shapes and Adaptive Scan’s rotating ability. This algorithm is the only one (in created simulator) that is capable of rotating tasks by 90°. This experiment was designed in such a way to show that
for a specific queue, it is possible to increase the overall efficiency of the algorithm a lot by rotating the tasks. It is shown in this experiment that the Adaptive Scan is the fastest algorithm because it can allocate more tasks in the same time compared to any other strategy.

Fourth experiment is similar to the previous one because the task dimensions are being changed. However, in this experiment the shape of the tasks in the queue is fixed and always square-like and the Adaptive Scan’s rotating ability was not an advantage anymore.

The last experiment changes the task processing time parameter. It is intuitive that for a queue with tasks that require more processing time, the whole simulation time - $T$ - will be higher. This experiment is also designed in such a way to show the problem that has been explained a few times earlier in this work. When the processing time is too short, the mesh fulfillment level is relatively low because of incoming deallocating requests. Such requests can keep coming before the whole network is scanned and they will 'block' the algorithm’s allocating possibilities.

In almost all cases, the Snail Algorithm proved to be the most efficient strategy. The Adaptive Scan is able to allocate certain tasks while other algorithms failed, thanks to its ability to rotate incoming tasks. Experiment three is designed in such a way to demonstrate situations when rotating tasks can really help to increase the algorithm efficiency.

Results of this research has many applications. It mostly depends on how much information we have about the problem (tasks in queue) and what the parameters of the system (available network) are. It is also important to know which parameters are the most important for the network owner in a specific case - average network utilization level or maybe simulation time? Maybe both of them? When the length
of the queue, task average dimensions and / or tasks processing time are known, it
gives enough knowledge to decide which algorithm should be used and what should
be the optimal network dimension. If it is possible to change the task parameters,
then they can be fine-tuned so that the tasks could be processed in a more efficient
way.
References


Bartosz Duszel
Stanisława Kunickiego 55B/3
Wrocław, 54-616
POLAND
+48 692-888-271
bartosz.duszel@gmail.com

EDUCATION

2012-2013: UNIVERSITY OF NEVADA, LAS VEGAS
Master of Science in Electrical Engineering

2011-2012: WROCŁAW UNIVERSITY OF TECHNOLOGY (master’s degree)
faculty of electronics, major: ADVANCED INFORMATICS AND CONTROL

2011-2012: WROCŁAW UNIVERSITY OF TECHNOLOGY (master’s degree)
faculty of electronics, major: ADVANCED INFORMATICS AND CONTROL

EXPERIENCE

Graduate Assistant - Teaching Assistant January 2012-December 2013
University of Nevada, Las Vegas Las Vegas, NV

• teaching introduction to engineering experience, digital logic I and II,
• working with the breadboards and different TTL chips,
• assembly basics for Nios II,
• implementing basic logic circuits.

DECT Tester August-October 2011
PGS Software - Gigaset Communications Wroclaw, Poland

• tested software and hardware on DECT terminals and base stations,
• learned how to work with CAFT.NET.

Tester (internship) July 2011
PGS Software Wroclaw, Poland

• tested websites and software on mobile devices,
• worked with ‘selenium’ tool,
• learned JIRA and SCRUM.

Wireless Technician 2009-2011
Wroclaw University of Technology Rover Team "SCORPIO" Wroclaw, Poland
- responsible for wireless connection with the robot and wireless video transmission,
- created promotional presentation, brochure and project website.

**Technician (half-time work)**

January-March 2011

Wroclaw University of Technology

- worked with MySQL database, searched for statistical dependancies,
- created simple PHP scripts connected with the database,
- together with supervisor presented formula describing weather changes over the year.

**Smarter Security (internship)**

December-March 2010-2011

IBM ESI

- learned rules of working in IBM corporation and dedicated IBM software for projects organization,
- learned about private cloud and web-applications security,
- worked with products from IBM Rational family.

**Internship (ended with honorable mention and award)**

June 2010

IBM mc²

- introduction to Smarter Planet conception and IBM corporation,
- learned DB2 fundamentals and passed few Proof of Technology workshops,
- passed "DB2 9 Database and Application Fundamentals" certification exam.

**Developer and Tester**

September-June 2009-2010

Wroclaw University of Technology

- participated in a government-sponsored research project,
- analyzed results of simulations and statistical data of the properties of a radio channel inside buildings and the reverberation chamber,
- developed scenarios for measurements and simulations of the impact of absorbing elements placed in a reverberation chamber on amplitude and temporal parameters of an electromagnetic field inside the reverberation chamber,
• developed additional module for main simulation environment application.

**ACTIVITY**

2013-2014: UNLV IEEE Student Branch
2009-2012: Wireless-Group

**COURSES**

2012: Gamification Course, Kevin Werbach (Associate Professor, University of Pennsylvania)
2010: IBM DB2 9.7 Academic Workshop
2010: IBM Rational AppScan Standard Edition v7.7

**CERTIFICATES, HONORS AND AWARDS**

University of Nevada, Las Vegas
Golden Key
Selected and nominated for membership in Golden Key - International Honour Society, Fall 2012.

University of Nevada, Las Vegas
The Best Teaching Assistant
Award for the best teaching assistant in Electrical and Computer Engineering Department, Spring 2012.

The Mars Society
Certificate of Participation
Certificate of participation in University Rover Challenge 2011 in the USA (4th place).

**CISCO**

Certified Network Associate (CCNA)
semester 1 (networking basics),
semester 3 (switching basics and intermediate routing).

**IBM**

DB2 9 Database and Application Fundamentals,
Educational Student Internship.

**ENGLISH**

ACERT
Academic Certificate, B2 level.

**SKILLS**

Operating Systems
Mac OS X (advanced),
Windows (advanced),
Linux, Unix (basics).

**Programming Languages:**
Objective-C (basics / intermediate),
C++ / C++11 (basics).

**Frameworks and Game Engines:**
Foundation, UIKit (basics),
cocos2D (basics),
Unity3D, UDK (basics).

**Rest:**
Git,
Xcode (intermediate),
TeX, LaTeX (intermediate),
JIRA, CAFT.NET, Matrix.Net,
SCRUM.

**LANGUAGES**
polish (native),
english (advanced).