TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY

Mat-2.177 Seminar on case studies in operation research

Performance Analysis of Distributed Computing in the Home Environment – An Explorative Study

Final report

19.4.2004

Nokia Research Center

Spring 2004

Pasi Kuusela, PM
Anton Danielsen
Henri Hytönen
Tero Ylä-Anttila

Summary

This study discusses the achievable benefits of distributed computing concept in home environment. The standpoint is in computational aspects and in relative speed-ups that could be obtained. This study has an explorative nature, and therefore recognizing fields of further studies have also been given weight.

To tackle the problem, two mathematical models (basic optimization approach and simulation approach) have been formed. Besides these, the upper bounds of obtainable speed-ups are approximated separately. Finally, results were compared by varying the computational requirement of the problem and the number of slave devices while other parameters were kept constant.

The results support the original assumption that computational problem should, in general, be split between devices according to their computational performance. This is especially the case, as the computational requirement of the problem increases causing the network to become less of a restricting factor. Thus, the utilization of the network becomes irrelevant for problems having large computational requirement. Packet size does not affect remarkably to the division ratios as long as the computational problem can be decomposed to sufficiently many distinct parts. All tested models imply that gained speed-ups are low for small scale problems.

Even though tests probably give overoptimistic results, it seems that speed-ups (i.e. time to compute on host solely per time to compute on all devises) of 10 could be achieved in typical home environment when the host device is typical modern mobile phone. On the other hand, if the central device is PC or laptop with relative large computational power, speed-ups tend to limit, based on tests, only up to 2.

However, models used in this study have their limitations and therefore reader should keep in mind that simplifications made can lead them far from reality in certain cases. Therefore project team recommends following fields of additional study: What is the affect of decomposition costs, how do multiple connection types in simulation affect in different network topologies and more realistic modeling of complexity and computational requirements.

Table of contents

1 Introduction

1.1 Background

Many computational applications in science and engineering require up to teraflops of computing capability [Kumm and Lea, 1994] and are therefore too time-consuming for even the most powerful ordinary computers. The rapid development of information networks has enabled the possibility to divide the problem to a number of smaller ones and then distributing these to independent computing nodes which can process smaller problems simultaneously. This kind of processing is known as Grid computing. According to Grid Computing Info Centre [GCIC], "Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements." Currently grid computing is used in many computationally demanding scientific experiments and also several public grid computing projects exist; the most well-known being the SETI@home project [SETI@home]. In many of these, the computational data is distributed through the Internet and the computing workload is divided between users who have voluntarily joined the project. These projects are all strong proofs that extremely challenging computational problems can be successfully handled and remarkable benefits gained if computing power several order of magnitudes greater would be available.

Grid computing in the home environment was the initial working topic for the project from Nokia. As our understanding of the issue grew this terminology was revised. The term Grid computing is commonly used for large networks of computation units and in which the availability of computing resources is not guaranteed. As Grid computing refers to a virtual pool of resources that the user has access to, while in our examined environment the computational resources are always available or the state of the system is at least known, the term Grid computing is somewhat inaccurate to describe this environment. Our study concentrates more on smaller networks, virtual pools of computational nodes that can all be accessed and all of the resources in the nodes can be

utilized by the user. Thus the environment examined is much closer to the qualities of conventional distributed computing environments than those of Grids.

During the recent years computational tasks have also proliferated in home environment. Mobile phone applications, digital photographing, computer and console gaming and digital videos, among others, can be mentioned. However, not all of these devices have as much computing power as normal PC does, though. Therefore interesting questions could be: If more computing power was available, would it allow new benefits concerning the applications on areas mentioned, and could the computing power of other devices be utilized in that case?

1.2 Objectives and Research Problem

The client's initial interest was to find out if it would be reasonable to utilize distributed computing in the home environment. Answering this question became the focal objective of the study. However, project group soon found out that the goal was vague and wide-ranging as such and needed to be adjusted. Objective and research questions were reworked and shaped with the client as the project proceeded and more insight was gained. Restructured research questions were derived from project's salient objective and are stated as follows:

1. *What kind of performance improvements can be achieved with distributed computing?*
2. *Which devices are essential in distributed computing in selected environments and problem types?*

As for explorative study, building framework and drawing conclusions is complemented by discovering the most essential fields for further research. Therefore an additional research question is:

3. *What are relevant fields to study this area further?*

1.3 Delimitations of the Study

This study is explorative and concentrates on computational aspects and performance analysis of distributed computing. In-depth dissection has been mostly omitted and several important fields needed to be delimited, among others the following ones:

- Applications and algorithmic issues are not covered.
- Detailed examinations of different topologies are omitted
- This study does not concentrate on sensitivity analysis
- Network's reliability analysis and fault tolerance of distributed computing networks are outside the scope of this study

1.4 Structure of the Study

The structure of the study is presented in *Figure 1*. With test runs and analysis, preliminary results related to achievable and measurable performance improvements (research question 1) and essential distributed computing network components (research question 2) are obtained. Applied models should emphasize different aspects and delimitations to support the finding of relevant fields of further studies (research question 3).

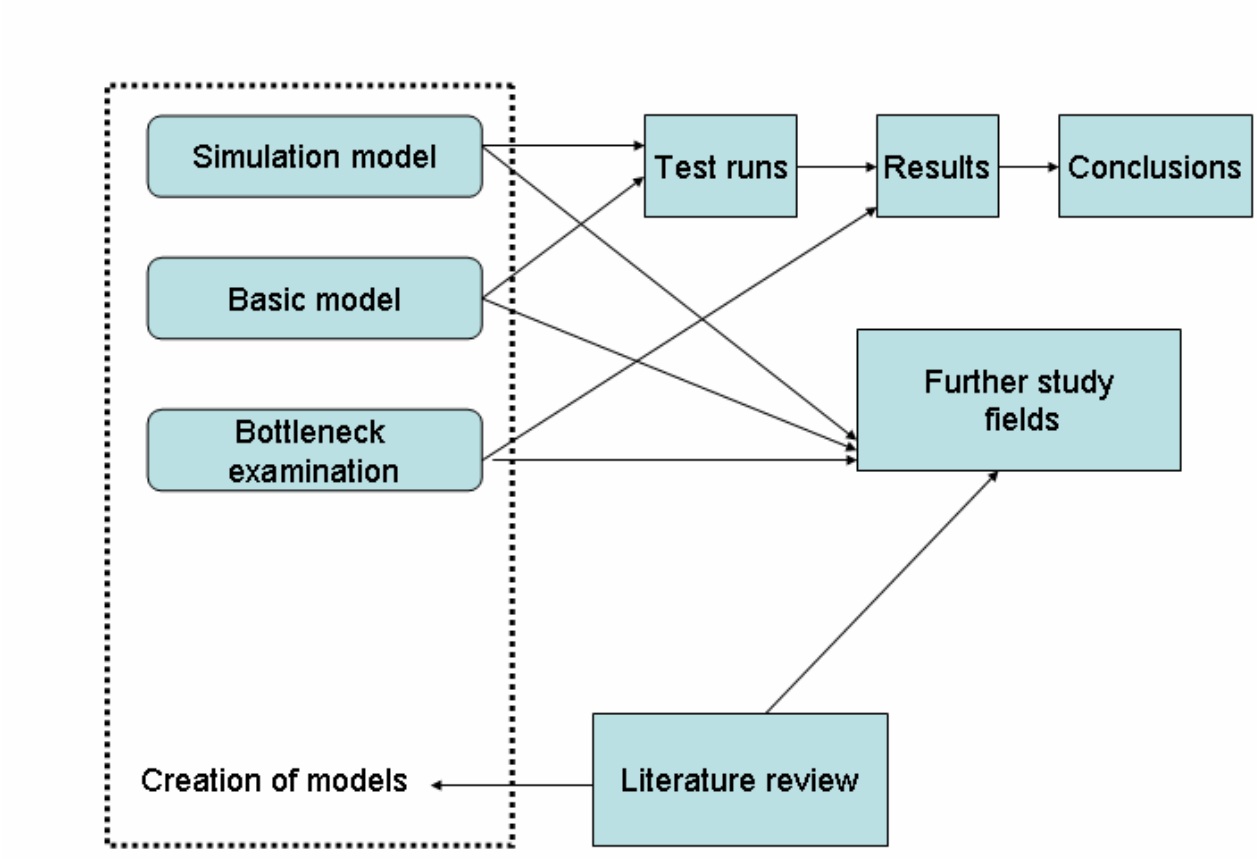

Figure 1. Structure of the study.

Even though not discussed separately, also a brief literature review has been conducted during the project. Its aim was to make the project team more conversant with distributed computing as well as support the creation of mathematical models. Third purpose was to avoid unnecessary work related to "refinding" further study fields.

1.5 Terminology

Perfectly decomposable problem is defined here – adapting the idea of the definition in [Thiébaut 1995] – as a problem that can be partitioned effectively (i.e. incurring no extra costs) to as many mutually independent sub-problems as is required.

Computing unit is a device that contributes in the computing process. These can be divided to two groups: **host devices** (root node of the network) and **surrounding devices** (leaf-nodes of the network). The latter ones are also referred here as slaves or terminal nodes.

Host device is the central computer in the network that is the only computer distributing the computing task to other computing devices in the network. **Master** is used as a synonym.

Grid Computing is defined here as any computing that involves splitting the computational task between two or more terminals. **Distributed computing** is used as a synonym in this study.

Network has a **star topology** if and only if it has n nodes, one being the root node and n-1 leaf nodes. Distance from any leaf node to the root node is exactly one unit length *Figure 2)*.

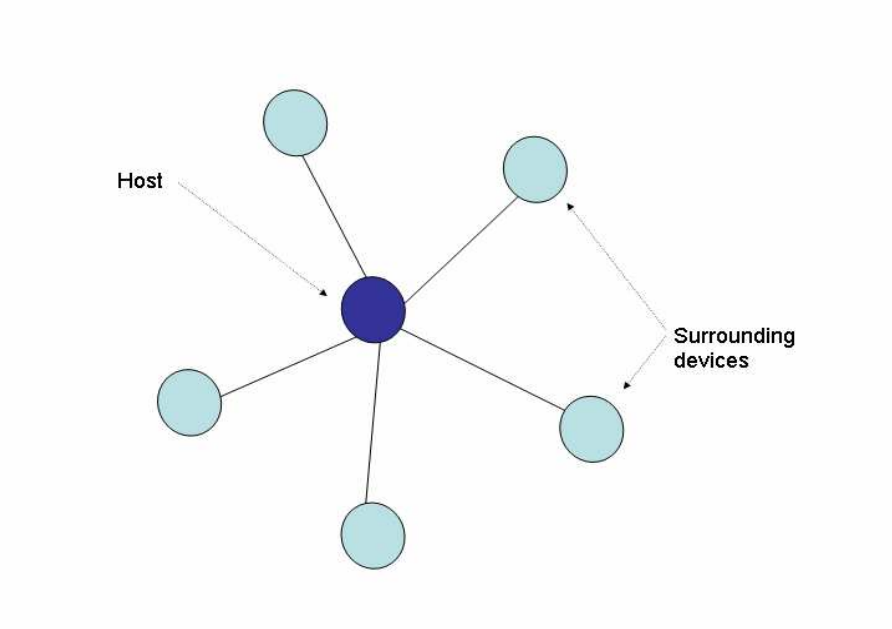

Figure 2. Star topology.

Environment is an element of the set *E*, which consists of all the different possible set-ups of computing devices that are connected with certain communication links in compliance with certain topology.

2 Technology overview

This chapter gives brief overview of technology related aspects of the study. It deals with technological background and the assumptions that have been made concerning the devices and types of connections between them. Also the selected environments are briefly described.

2.1 Computing Devices

Contemplation of computational devices has been limited to the following ones: Personal computer, laptop, mobile phone, Console, PDA and DVB-T set-top box. These devices have been selected for having significant computing power and being an integral part of hardware of a modern home. As the timely focus of this study is in medium term (2+ years), it is probable that these devices are in large use then and have a status of commodity utilities.

Computing power of each device is assumed to be directly proportional to the speed of each CPU when measured in megahertz. Thus the computing power describes the number of basic computational operations the processor can perform per unit time. The impact of all the different processor architectures and designs is ignored. For example, it is known that AMD Athlon XP 3200+ (nominal CPU speed of 2200 MHz) is roughly equal to Intel Pentium 4 2600 MHz [Tom's Hardware Guide 1].

Assumed computing powers for the computing units have been listed in *Table 1*.

Table 1. Computing powers of computing units.

Unit	*Computing power*	*Source*
PC	2 800 MHz	Uni Pentium 4 desktop SFF workstation [IBM 1]
Laptop	2 400 MHz	IBM ThinkPad G40 [IBM 2]
BT mobile phone	104 MHz	Nokia 6600 [InfoSyncWorld]
WLAN mobile phone	300 MHz	Samsung SGH-i700 [InfoSyncWorld]
PDA	400 MHz	Palm Tungsten C
Console	733 MHz	Microsoft X-Box
DVB-T box	166 MHz	Nokia Mediamaster 260s [Nokia]

2.2 Communication links

Each communication link has a capacity, certain rate it is capable of transferring data.

In theory, **Bluetooth** can transfer data as fast as 125kB/s [Phan]. It is assumed that the speed of BT is **70kB/s**, which appeared to be the realistic capacity in experiment where several MPEG-4 files were transmitted over Bluetooth connection [Chia].

Bluetooth link has the following operational principle: Every new Bluetooth connection creates a piconet, which has one master and maximum of seven slaves. Master can send data to every slave simultaneously, but each slave reserves whole bandwidth when sending data back to the master. This means in practice, as each data packet sent to slaves is different, that only one master-slave pair can communicate at the time. [Forum Nokia]

Wireless LAN (IEEE 802.11b standard) can theoretically transfer data at the speed of 1 441 792B/s [Phan]. In practice, packet errors due to non-optimal connection quality make the real transmission rate lower. Sprague estimates the real throughput to be between 790kB/s and 920kB/s [Sprague]. Therefore the value of **850kB/s** is used in this study.

Local Area Networks have usually data transmission capacity of 1,31MB/s (standard Ethernet IEEE 802.3) or **13.1MB/s** (Fast Ethernet IEEE 802.3u). The latter one is assumed to be in use. As Fast Ethernet can transfer the data at relatively high speed, other parts of the system usually become bottlenecks. Though, to keep mathematical models simple, we have elided this performance reducing effect.

Full speed **USB 1.1** (Universal Serial Bus) has a data transmission capacity of 1,5MB/s [usb.org], which is theoretical value. In practice, the transmission speeds are lower, usually between 112kB/s and the theoretical maximum value that can be achieved in multiple end point transmissions [usb.org]. In this study USB has assumed to have a capacity of **1,5MB/s**. This and other transmission rates have been aggregated in *Table 2*.

Each communication link has also a **latency time**, which is the time that occurs in every transmission regardless of the size of the packet transmitted. Latency is caused mainly by the need to condition the data before and after the transmission. Also the transfer itself increases the latency since the communication channel has certain length that the transferred data needs to travel with finite speed. The latencies of communication channels

are also listed in *Table 2*. The values are only rough approximations because they are highly dependent on the topology and number of devices among others and thus no single value to match every situation can be given.

Table 2. Transmission rates of different communication links. These values were also used in testing.

Link	*Transmission rate (Bytes per second)*	*Latency*
Bluetooth	71 680	50 ms [Forum Nokia]
Wireless Local Area Network (WLAN)	870 400	20 ms [Lunheim et al., page 3]
Local Area Network (LAN)	13 107 200	20-30 ms [Infopen Archivum]
USB	1 572 864	

2.3 Terminal Nodes' Connectivity and Topologies

Each surrounding device could have one or more communication links between it and the host device. For example, PC and laptop can be connected with WLAN, Bluetooth, LAN or USB. The available communication links are presented below in *Table 3*.

Table 3. Available communication links between terminal nodes (marked with X) and environments where those links are used (marked with A, B, C or D).

	Bluetooth	*WLAN*	*LAN (IEEE 802.3u)*	*USB 1.1*
PC	X (B,D)	X (C,D)	X (D)	X (D)
Laptop	X	X (D)	X	X
PDA	X (D)	-	-	X (D)
Mobile Phone	X (A,B,C)	X (C,D)	-	-
Console	-	-	X (D)	-
DigiBox	X (D)	-	-	-

It is assumed that each surrounding device has only one primary connection (i.e. the one that is most often used in normal situation) which is the only connection in use. In other words, it is assumed that there exists exactly one communication link between the host and each surrounding device.

Topologies are considered at the basic level. Most common ways to connect devices together are *bus*, *star* and *ring* topology. In bus topology, all devices are connected in series. Ring topology is in question when devices are connected in series and the first one and last one in series are also directly connected to each other.

2.4 Environments

Different computational devices, communication links and network topologies offer various ways to form environments where distributed computing concept can be dissected. We have selected four distinct home environment settings where analysis of developed models will be made. These settings (selected environments) are:

A. Two to eight similar cellular Bluetooth phones connected in a piconet

B. A mobile phone connected to one or several similar PC computers

C. One PC connected to one to several similar cellular phones through WLAN.

D. A star network with one PC connected to various other devices

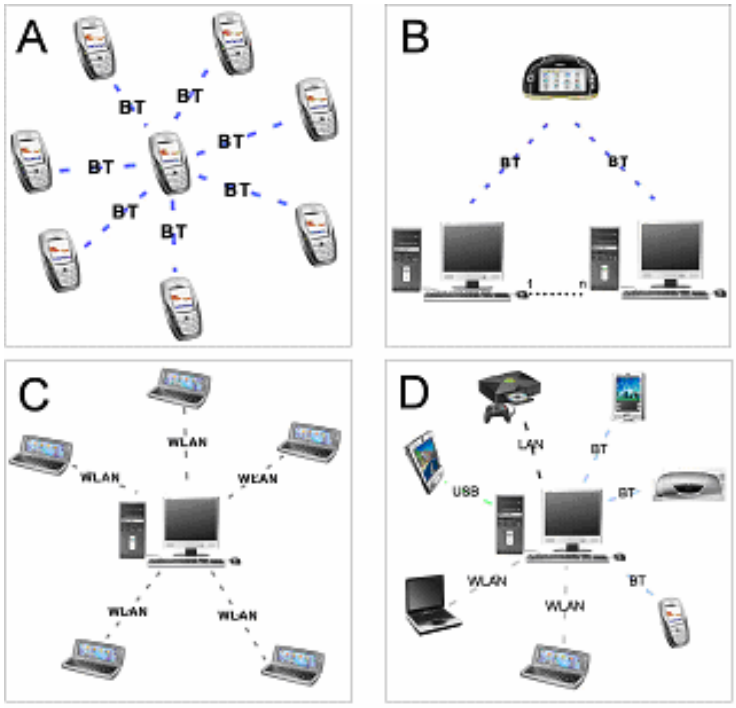

Figure 3. Four home settings of computing units. In the first setting (A) the problem host is cellular Bluetooth phone that has formed a piconet. In the second setting (B) the problem host is a cellular phone that communicates through a Bluetooth channel. In the third setting (C) a PC distributes its problem though a WLAN to mobile phones. In the last setting (D) different devices from a home environment provide computing to a PC communicating though various channels.

All network topologies are star networks *(Figure 3)*. The device with a problem to be solved is illustrated in the center of the network and is designated as the problem host device or master in this paper. The surrounding devices are processing units that participate in solving the problem by providing computing power.

One simplification that has been made is that no data collision exists and the frequencies of different types of wireless communication links do not have any interference. This is not completely realistic because - as Urosevich argues – Bluetooth can destroy WLAN connection (IEEE 802.11b) they are "even remotely coming into contact with each other" [Urosevich]. All the necessary middleware software is assumed to be available and incurring no additional costs to computing.

3 Mathematical model

In this study, different types of models are used. Besides emphasizing different aspects to consider in distributed computing, models have another task. The **basic model** is used to give the lower bound for the avail that can be achieved using distributed computing for various computational problems. The **bottleneck examination** is used to approximate the upper bound of available benefit. The **simulation approach** is used to approximate the most likely benefit obtainable from distributed computing.

3.1 Variables and Parameters

In this chapter we introduce the key variable and parameters by which we describe the grid computing domain. A complete list of the variables and parameters used in this document can be found in the appendix A.

3.1.1 The grid environment

The network has a star topology *(Figure 2)* and consists of m+1 computing units, with the central device corresponding to index zero and the surrounding devices corresponding to indices 1,…,m.

The time taken to perform the computation of a problem with computational requirement y on the i^{th} device that has a processing power of c_i is defined as

$$t_{compute} = \frac{y}{c_i} \quad i=1,\ldots,m. \tag{1}$$

Between the central device and each surrounding device it is assumed that there exists a symmetric communication link. The time taken for a file of size x to move to or from the i^{th} device is defined as

$$t_{transmit} = \frac{x}{s}, \tag{2}$$

where s is the capacity denoted to transferring the file. Equation (2) is a simplification of that presented by Culler et al [Culler et al. 1993]. In their paper the total message communication time of a x bit long message was given by

$$T(x,h) = o_{snd} + \left\lceil \frac{x}{s} \right\rceil + hl + o_{rcv} \quad (3)$$

where o_{snd} is the send overhead. This is the time that the processor is busy interfacing to the network before the first bit of data is placed into the network. s is the bandwidth of the channel. h is the number of hops a message must across before arriving to the destination and l is the length of each hop. Finally, o_{rcv} is the receive overhead, the time from the arrival of the last bit until the receiving processor can do something useful with the message. Some examples of capacities and channel lengths (latencies) can be found in *Table 2*. In the network examples presented by Culler et al. the send and receive overheads were together smaller than one ms, although these networks were dissimilar to the ones of this study. Thus, the same overheads do not inevitably apply. Still, applying (3) to a case, in which a 100 kB message is sent over Bluetooth channel with capacity 723.3 kbps and latency 50 ms, and assuming overheads are 1 ms, the time required to send a message would consist of overheads of 0,001 seconds, latency of 0,05 seconds and 1,1 seconds to push the message into the channel. Hence, equation (2) is not far from equation (3), when package sizes are large (larger than 100 kB in Bluetooth networks).

3.1.2 The Task

The task is to compute a problem P by either solving it on the host device or by distributing it into the computing network. To distribute problem P, it must first be decomposed into n sub problems p_i. Decomposition is performed on the host device only. Sub problems (p_1,…,p_n) are solved by sending each one to some computing device located in the computing environment. Based on the answers of all p_i the final answer to P is composed. The time taken to solve the problem on the host device is denoted by $T_{sequential}$.

The time required to decompose the problem is denoted by $T_{decompose}$, the time taken to solve the set of sub problems is denoted by T_{solve}, and the time taken to compose an answer is denoted by $T_{compose}$. The total time of solving P by distributing it is denoted by

$T_{distributed}$, and is naturally based on the previously mentioned durations $T_{decompose}$, T_{solve} and $T_{compose}$. Still, $T_{distributed}$ need not to be the sum of $T_{decompose}$, T_{solve} and $T_{distributed}$, since the decomposition, solving and composition can happen partly in parallel.

3.1.3 Problem characteristics

The problem P to be distributed is characterized by its computational workload Y, its description, which has a size X in bytes (later referred to as size or description merely), and its answer, which has a size R in bytes (later referred to as answer merely). Similarly, each sub problem p_i has workload y_i, description x_i and answer r_i. If problem P can be divided into wanton independent parts p_i, such that

$$Y = \sum_{i=0}^{n} y_i \tag{4}$$

no extra costs will be incurred with the decomposition or composition, we call P perfectly decomposable. In the development of the mathematical models (chapter 3.2) it is assumed that the problem is perfectly decomposable, and thus it can be concentrated on modeling the cost $T_{distributed}$. The solutions should still be applicable to situations in which decomposition and composition can be performed in parallel with other tasks, thus not increasing the total time, or by small modification to situations in which the decomposition and composition costs are fixed.

3.1.4 Performance metric

As performance metric, gain or relative speed-up G, resulting from distributing the problem, is used. The gain is defined as

$$G = \frac{T_{sequential}}{T_{distributed}}. \tag{5}$$

The same definition is also widely used in literature (see Kumm et al., for example).

3.2 Basic Model

3.2.1 Assumptions

In the basic model we assume the problem P is decomposed into as many parts as there are computing devices in the grid and the i^{th} problem is sent to the j^{th} computing device (j is equal to i-1) . Hence, the amount of problems n is equal to the amount of computing devices m+1 in the grid. Furthermore, we assume that the decomposition is completely performed before solving the set of sub problems $\{p_1,\ldots,p_n\}$ is started. For each sub problem p_i we assume that the solving consists of the discrete steps of sending the problem description x_i to the j^{th} device, solving the sub problem p_i on the j^{th} device and sending back the answer a_i of the i^{th} sub problem to the host. Finally, we assume the complete set of sub problem answers $\{a_1,\ldots,a_n\}$ must arrive at the host before the problem can be composed and said to be completed.

3.2.2 Derivation

The objective is naturally to minimize the time taken to solve problem P in the grid environment. Based on the assumptions, the time taken to solve the complete problem P_i depends on the individual solving durations t_i in the following way

$$T_{solve} = \max(t_i) \tag{6}$$

Based on the assumption the solving time of the first problem, computed on the host device takes

$$t_1 = \frac{y_1}{c_0} \tag{7}$$

and the solving time on the surrounding devices takes

$$t_i = \frac{x_i}{s_j} + \frac{y_i}{c_j} + \frac{r_i}{s_j} \tag{8}$$

The idea is that we may split the total amount of calculations into $n = m+1$ sub problems in any desired way restricted only by equation (4). Also the capacity is split among devices using the same network. This is done independent of time. Thus, if one device is devoted s_i capacity it will have s_i capacity in use during the whole distributed computing process. Because devices using one channel split the total capacity of that channel we have a restriction as follows. If, for example, the first k devices use the same WLAN connection and the WLAN capacity is s_{WLAN}, then $s_1,\ldots, s_k$ are restricted by

$$\sum_{i=1}^{k} s_i \leq s_{WLAN} \tag{9}$$

Consequently, the decision variables are y_i:s and s_i:s. Before the model can be solved we have to make some assumptions about the description sizes x_i and the answer sizes r_i. We assume there is a function κ, which gives x based on y in the following way.

$$x = \kappa(y) \tag{10}$$

Additionally, we get r from

$$r = \beta x_i \tag{11}$$

Now, we can write the optimization task as

$$\begin{aligned}
&\min(T_{solve}) \\
&st. \\
&\quad \frac{y_i}{c_j} \leq T_{solve} \\
&\quad \frac{\kappa(y_i)}{s_j} + \frac{y_i}{c_j} + \frac{\beta \cdot \kappa(y_i)}{s_j} \leq T_{solve} \qquad (12) \\
&\quad Y = \sum_{i=0}^{n} y_i \\
&\quad \sum s_i \leq s_{BT} \qquad (one\ \ for\ each\ channel) \\
&y_i \geq 0,\ s_i \geq 0
\end{aligned}$$

3.3 Bottleneck examination

The purpose of the examination is to give upper bound for the benefit that can be achieved with the distributed computing concept. It is assumed that each surrounding device can start computing as soon as the first bit of the sub problem is transferred. Transmission of the data is possible to any number of communication links simultaneously as long as the total capacity of communication links is not exceeded. Bandwidth sharing is assumed to not to cause any extra costs. Data collision and other errors as well as the capacity taken by error corrections have been omitted and no queuing delays are assumed to exist. Some of these assumptions are partly unrealistic but the examination is mainly used in predicting an upper bound for the relative speed-up achievable.

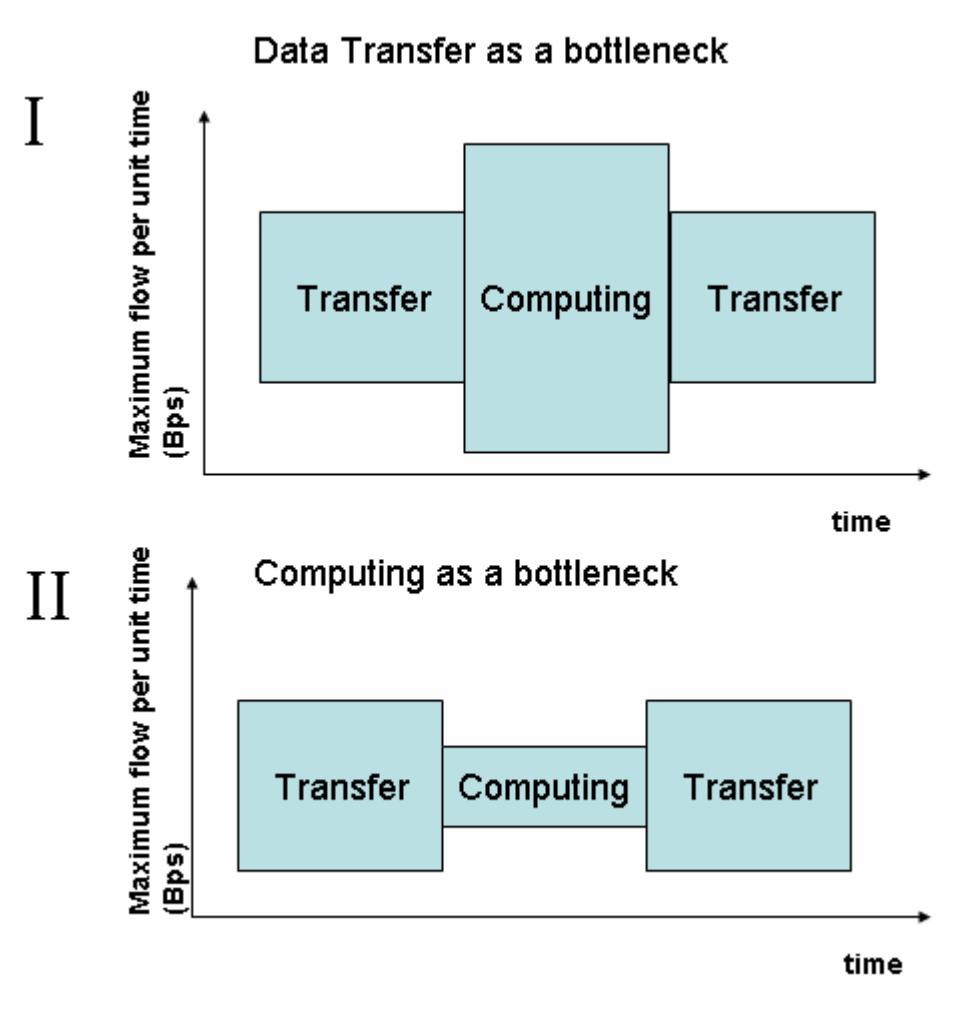

Figure 4. Bottleneck examination

The principle of bottleneck examination is presented in *Figure 4*. Either the available transfer capacity (case I) or the computing power (case II) forms a bottleneck. The transfer capacity is measured as the total bandwidth of all communication links (Bps) and the computing power as operations per second (approximated with CPU's speed in MHz). These two measures are linked together with complexity coefficient *k*. This is simplification of the equation (10) as the linear dependence between problem description and its computational requirement is assumed.

3.4 Simulation approach

Other type of problems besides those described in chapter 3.2, are the ones in which problem cannot be divided to arbitrary parts but instead the problem consists of smaller, independent sub problems of fixed size. This type of distributed computing paradigm is called bag-of-tasks, described for example by Kuang et al. [Kuang et al., 1999]. Bag-of-tasks type problems arise for example when same algorithm or function is to be applied to a wide set of different parameters. In typical bag-of-tasks type problems, different sub problems are not needed to be computed in any particular order. As long as there are tasks to compute, one task at a time is picked from the "bag" of remaining problems and computed.

3.4.1 Simulation model

To measure gains of distributed computing in the home environment when bag-of-task type problems are examined, a simple simulation model was created. The model, presented in *Figure 5*, is based on single-server type service centers for computation and transmission, each of which can serve only one task at a time [Mak and Lundstrom, 1990]. The computational problem consists of sub problems entering the system illustrated by the **incoming flow** in *Figure 5*. The size of each sub problem is assumed to be normally distributed.

Each sub problem must be solved by either the host or some surrounding device. If the problem is solved by the host, no transfer costs occur. Therefore the sub problem is immediately sent to host's **computing queue** that follows FCFS (First Come First Served) principle - as do all the queues in this simulation model. After host has accomplished the

given computational task, the solved sub problem enters the **set of solved sub problems**. It is assumed that there are not any assembly costs and thus the time taken to solve the problem is in this case simply t_{comp}.

If the single task is solved by some surrounding device, the situation is bit more complicated. The task is first copied to the system's **transfer queue**. It can be sent to any of the n surrounding devices. Assumption is that only one transfer can take place at the time. Sub problems are transmitted one by one to the computing queues of the surrounding devices.

The communication link has specific bandwidth and sub problem description certain size, which both affect the transfer time t_{trans}. Each surrounding device has its own computing queue. After the sub problems have been solved, they return to the transfer queue, which therefore contains both sub problems to be solved and solved sub problems to be transmitted back to the host. When solved sub problems reach the beginning of the transfer queue they are transferred back to the host computer and then copied to the set of solved sub problems. Thus also $t_{receive}$ depends both on the bandwidth and size of the sub problem as well as the content of the transfer queue.

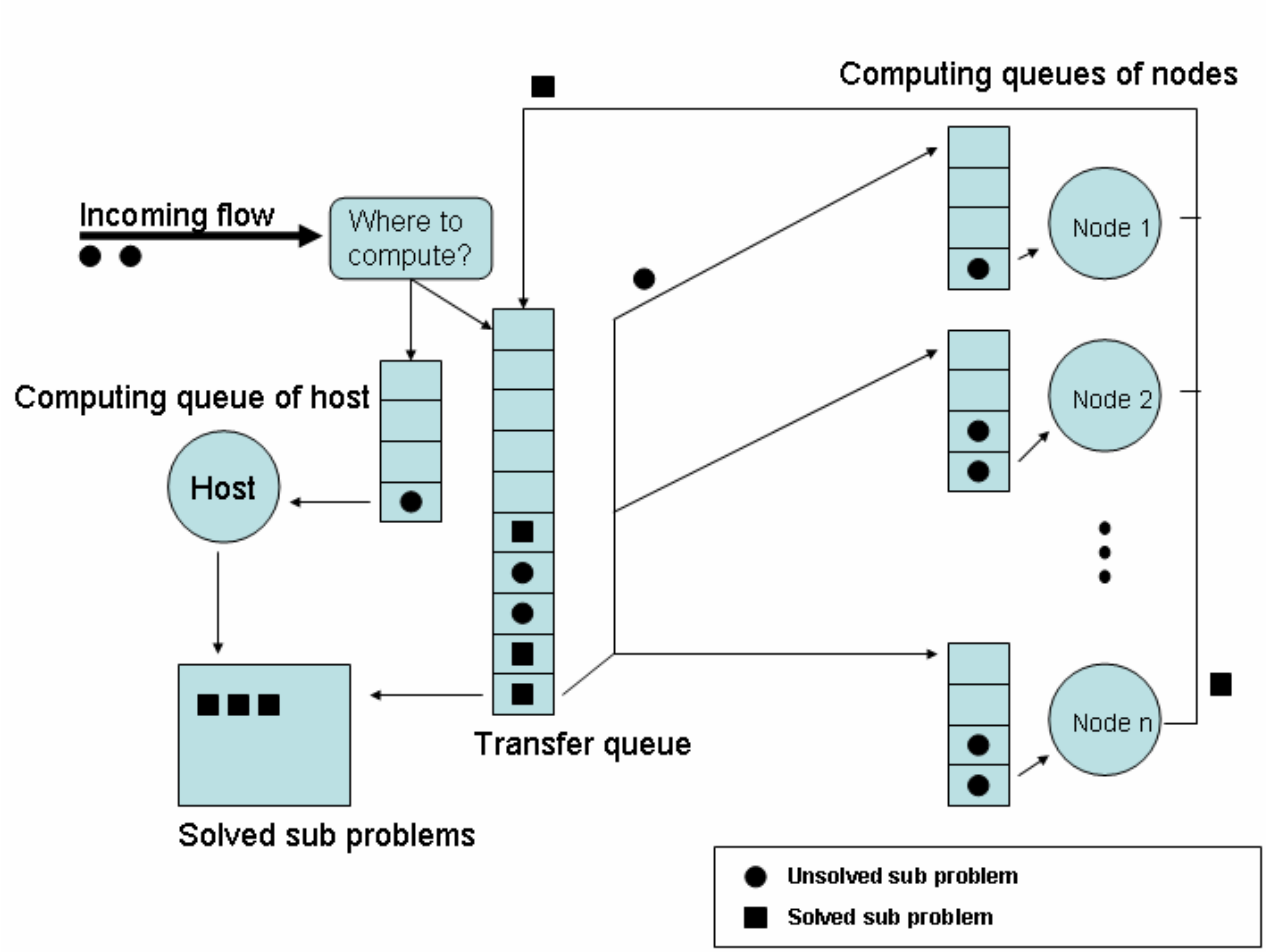

Figure 5. Structure of the simulation model.

The decision to which surrounding device the sub problem is sent, is based only on the current situation (fastest one of the nodes' computing queues is selected). In other words, the known information related to the distribution of new packets arriving to the system after the packet in question, is not exploited in decision making.

The simulation model is limited to the situation where there is only one type of symmetric communication link in use. Thus the use of simulation model is therefore restricted to the environments A, B and C. The constructed simulation model is implemented with MathWorks' MATLAB software. The source code of the simulation program is presented in appendix D.

3.4.2 Simulation procedure

The simulation procedure was carried out so that the environments A-C were tested with two sizes of problems, approximately 10MB and 100MB. The problems were divided into 50 and 500 fixed size packages of the size of X kB, X~N(20, 200). The problem represents a signal processing problem type without compression, so the problem packages transmitted to slaves and returned computed answers are of the exactly same size. The simulation model allows this to be altered, but in our opinion this was a good basic problem to take a deeper look into. One good reason to choose this problem type was that our previous empirical study had resulted complexity parameter k for video conversion. This parameter gives an estimate of how many computation operations is needed to convert 1 byte of input data.

Other fixed parameters associated with this particular simulation were that the actual speed of Bluetooth was set as 70kB/s and the speed of WLAN as 850kB/s. The environments A, B and C consisted of mobile phones, with a CPU speed of 300MHz and PCs, with a CPU speed of 3,2GHz (a common high performance CPU).

With simulation the differences between the computation times of distributed computation and the time of computation with the sole host machine were compared. The relative speed-up G and the division ratio of all the system's devices with respective computing environments were both calculated. This was done for both problem sizes for two reasons: first of all, to double check the simulation procedure, and secondly to compare the ratios for possible benefits of distribution related to computation problem sizes.

4 Results

Test runs were made in different environments utilizing the various models presented previously (basic, upper bound, and simulation). To give an idea of how various devices can contribute to the computation in home environments, the presentation of results is started with a simple example from environment D, where a particular fixed problem is distributed exploiting the basic model *(chapter 4.1)*. Thereafter, the other environments A, B and C are examined keeping the environment fixed, the problem fixed and making use of simulation *(chapter 4.2)*.

In *chapter 4.3* we will examine all three models concurrently. Contrary to preceding chapters the problem and environments are not kept fixed, as the computational requirement and the number of slaves in each environment are varied. The impact of different models, environments and problems is depicted with the relative speed-up.

4.1 Results of the Basic Model

Here results are presented from a fixed environment D and a fixed problem description size of 10 MB and the complexity coefficient k of 10,8 k/B. Computational requirement and sequential solving time of the problem were derived using these mentioned values. The environment and the problem are summarized in the last column of *Table 4*. The results from this setting are presented in *Figure 6*. The left hand side of *Figure 6* shows in what proportion the problem is distributed to the devices in the environment and the right hand side shows as a comparison the distribution of the computing powers of the devices.

Although the distribution of the problem reminded the distribution of computing powers there existed also significant differences. PC as the master, laptop and Game console are in dominant roles computing 88 percent of the task, a little more than their share of the total computing power. Other devices such as hand-helds and the DVB box contribute only marginally and do not offer remarkable speed-up per se, even if they are equipped with fast communication link.

The master PC performed nine percentage points more of the total computing than its share of the total computing power is. This is due to the fact that the master device has more computational time compared to slaves, as there is no need to perform transfer of the problem over some communication link. Similarly, except for the game console, all slaves did perform a smaller slice of the total computation than their share of the total computational power of the network, because the transfer of each sub problem over the communication link consumes part of their possible computational time. For example, the laptop received eight percentage points less of the problem to compute than its share of the total computing power was. On the other hand, the game console received more of the computational problem than its share of the computational power was. This was because the game console had the clearly fastest communication link (and did not share it). In other words, although the computing power to a great extent guided the distribution, each communication link's capacity had also some significance. The situation would have been different had the complexity coefficient been other.

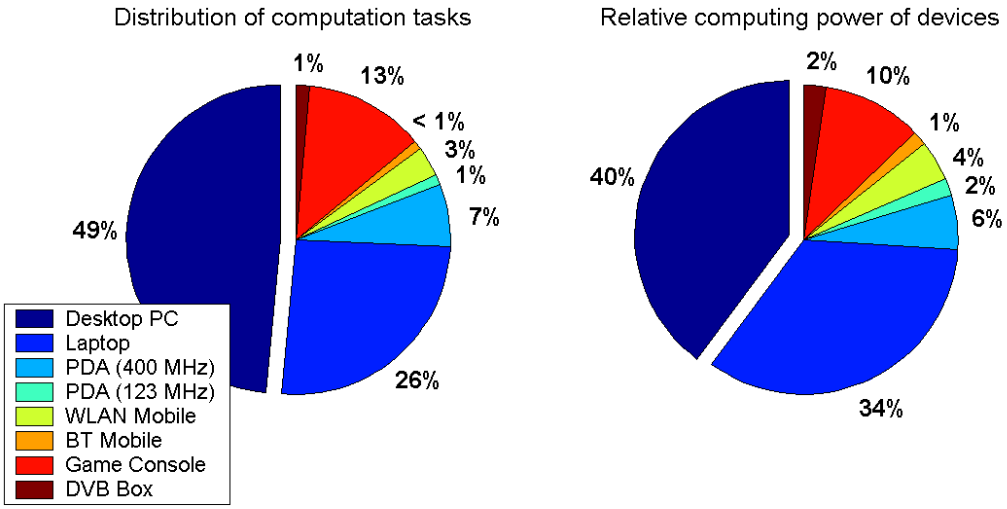

Figure 6. Distribution of computation tasks with the basic model in the environment D and relative computing power of different devices in environment D.

4.2 Simulation results and analysis

It was observed that the changes in division ratios between 10MB and 100MB problem sizes were marginal. The settings of the environment and the problem can be seen in *Table 4* and the results from the test run are presented in *Table 5*.

Table 4. Environment set-ups that were used in the test of the basic model.

	Environment A	Environment B	Environment C	Environment D
Master device	Bluetooth mobile phone (104 MHz)	WLAN mobile phone (300 MHz)	Desktop PC (2,8 GHz)	Desktop PC (2,8 GHz)
Slave devices	4 Bluetooth mobile phones (104 MHz)	Desktop PC (2,8 GHz)	7 WLAN mobile phones (300 MHz)	Laptop PC (2,4 GHz) PDA (400 MHz) PDA (123 MHz) WLAN phone (300 MHz) BT phone (104 MHz) Game Console (733 MHz) DVB box (166 MHz)
Communication channel(s)	Bluetooth (560 kbps)	WLAN (6,5 Mbps)	WLAN (6,5 Mbps)	Bluetooth (560 kbps) WLAN (6,5 Mbps) USB (12 Mbps) LAN (100 Mbps)
Size (X) [MB]	10/100	10/100	10/100	10
Sequential solving time (Tsequential) [min]	18,1	6,3	0,7	0,7
Complexity coefficient (k) [Hz*s/B * 10^3]	10,8	10,8	10,8	10,8
Computational requirement (Y) [Hz*s*10^9]	113	113	113	113

Table 5. Simulation results

	Distributed computation				***Sole host***	
Environment A	10MB Problem		100 MB Problem		10MB Problem	100 MB Problem
Total computing time	216,5s		2165,2s		583,6	5941,1
Relative speed-up	2,7		2,7			
	Composition of Environment A					
Devices	**Mobile 0 (host)**	**Mobile 1**	**Mobile 2**	**Mobile 3**		
Division ratios	36%	22%	22%	20%		
Environment B	10MB Problem		100 MB Problem		10MB Problem	100 MB Problem
Total computing time	204,7s		1990,4s		583,6	5941,1
Relative speed-up	2,9		3			
	Composition of Environment B					

Devices	**Mobile 0 (host)**	**PC 1**		
Division ratios	32%	68%		
Environment C	10MB Problem	100 MB Problem	10MB Problem	100 MB Problem
Total computing time	63,7s	620,5s	98,7	1006,2
Relative speed-up	1,5	1,6		
	Composition of Environment C			
Devices	**PC 0 (host)**	**Mobile 1**	**Mobile 2**	**Mobile 3**	**Mobile 4**
Division ratios	60%	10%	10%	10%	10%

The simulation analysis clearly shows the fact that even in the case of sequential transmission and over a relatively slow Bluetooth connection it is worthwhile to divide the problems to slave devices. This does not necessarily apply to the problems that are time consuming to decompose, but in this case of signal processing it is very fast and straightforward procedure. In the Bluetooth connected environments the bottleneck for distributed computation is the connection speed and in WLAN environment it is the processing speed.

Model gives surely slightly over positive results, relative speed-ups being around 3 in the Bluetooth environments and around 1,5 in the WLAN environment (C), for the latter the reason being quite simply the fact that in CPU speed the PC matches the CPU speeds of roughly 9 mobile phones. It can still be observed that the current high-end processing devices used at home are – and can be used as powerful tools for computing complex and time consuming problems.

4.3 Comparison of simulation, basic model and bottleneck examination results

In this chapter results from runs with all three models (basic model, bottleneck examination and simulation) are presented. Two separate simulations were done in each examination, one in which the problem was split into 10 sub problems of equal sizes, and another one in which the problem was split into 50 equally sized sub problems. Thus, results are presented from altogether four approaches. The first part of the chapter considers how relative speed-ups change when the computational requirement of the

problem varies. And the second part considers how relative speed-ups change when the environment varies.

4.3.1 Varying computational requirement

In this part results from fixing the environment and varying the computational requirement of the problem are presented. This was done by first fixing the problem description size X at 20 MB, 60 MB, 600 MB and 600 MB in the environments A, B, C and D, respectively. The problem was to perform a task on this file, which takes anywhere from 5 to 120 minutes to perform when the computation is solely made on the master device, i.e. $T_{sequential}$ is in the interval [5 min, 120 min]. The file could, for example, be a video clip and the task is some conversion process that is performed on this file. A five minute conversion corresponds to the problem having the least computational requirement and the two hour conversion corresponds to the problem having the highest computational requirement.

In *Table 6* the environments and problems are summarized. The complexity coefficient k and the computational requirement Y were derived in each environment from the computational power of the master device, the description size and the duration of solving the problem on the master device solely ($T_{sequential}$). As both the description and computational power of the master device triples and nearly triples respectively when moving from environment A to B, the complexity coefficient interval is nearly the same both in environment A and B. The same applies when going from environment B to C or D, as the description size and the computational power of the master in environments C and D are about ten times those in environment B. Thus the complexity coefficient stays nearly constant from environment to environment.

Table 6. The environments and computational problems.

	Environment A	Environment B	Environment C	Environment D
Master device	Bluetooth mobile phone (104 MHz)	WLAN mobile phone (300 MHz)	Desktop PC (2,8 GHz)	Desktop PC (2,8 GHz)
Slave devices	4 Bluetooth mobile phones (104 MHz)	Desktop PC (2,8 GHz)	7 WLAN mobile phones (300 MHz)	Laptop PC (2,4 GHz) PDA (400 MHz) PDA (123 MHz) WLAN phone (300 MHz) BT phone (104 MHz) Game Console (733 MHz) Digibox (166 MHz)
Communication channel(s)	Bluetooth (560 kbps)	WLAN (6,5 Mbps)	WLAN (6,5 Mbps)	Bluetooth (560 kbps) WLAN (6,5 Mbps) USB (12 Mbps) LAN (100 Mbps)

Size (X) [MB]	20	60	600	600
Sequential solving time (Tsequential) [min]	5-120	5-120	5-120	5-120
Complexity coefficient (k) [Hz*s/B * 10^3]	1,49-35,7	1,43-34,3	1,34-32,0	1,34-32,0
Computational requirement (Y) [Hz*s*10^9]	31,2-749	90,0-2160	840-20200	840-20200

The results of the runs can be seen in *Figure 7*. All models provide awfully low speed-ups when the duration $T_{sequential}$ was set at 5 minutes. As $T_{sequential}$ was increased speed-ups also increased. When examining the upper bound curves in all environments, the following can be perceived: At 5 minutes speed-ups are at about 1.5, 2.9, 1.2 and 1.6 in environments A, B, C and D, respectively. The limiting factor is the network transmission speed. As the computational requirement of the problem increases the network becomes less of a restricting factor. At 20 to about 40 the upper bound of computational power is reached. Thus, thereafter the restricting factor is the computational power of the slave devices and no more speed-up is achieved when the computational requirement is increased from that point on. The upper bound plateau settles at the sum of all computing devices in the network divided by the computational power of the master.

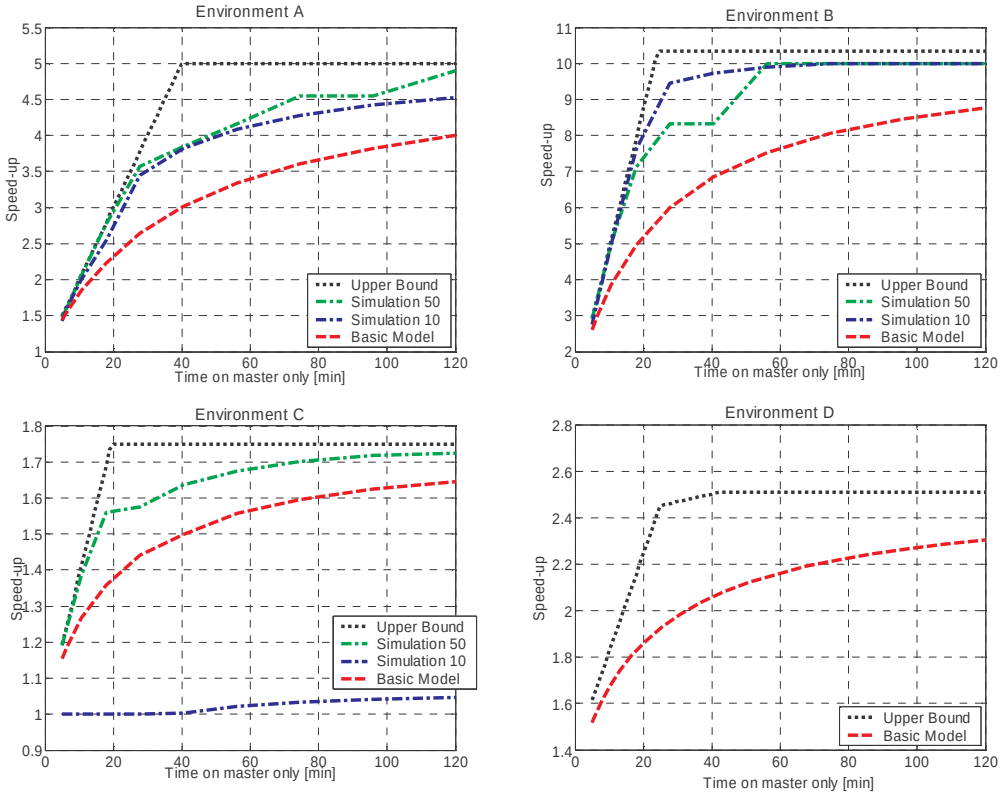

Figure 7. Relative speed-ups as a function of the computational requirement in the environments A, B, C and D. The x-axis shows how many minutes it would take to solve the problem on the master

device only, and the y-axis the relative speed-up. The problem parameters are summarized in *Table 6.*

All the other models move beneath the upper bound, as naturally should be the case. Furthermore, they all start at about the same starting point and converge to the upper limit as the computational requirement increases. The reason for the difference between the upper bound and the other models is that there existed moments during the runs when either the network was idle and/or the computing devices were idle in both the basic model and the simulation models runs. The idle time is intuitively clear from the basic model, because the slave devices do no computation during transfers over the network (and vice versa). In all environments at least one of the simulation models outperformed the basic model. This is because there was more idle time in the basic model solutions. In other words, in the simulation runs the network and/or the computing devices were kept to a greater extent busy. For example, in the simulations the first slave may start computing already at the point when the first assignment arrives, which arrives much sooner in simulations than in the basic model since the packages are smaller and the network is devoted to sending the first package only. Furthermore, as the device solves the first sub problem it may concurrently receive more sub problems over the network. Also it is worth noting that all models keep increasing somewhat although the upper bound has reached its plateau. Partly as a consequence, the difference between the upper bound speed-up and the speed-up of other models seems to be the greatest at the bottleneck turning-point. As the computational requirement increases to infinity the network transmissions duration become insignificant in comparison to the computational durations. Thus, how the network is utilized is not so important, and furthermore the problems is split between the different devices more and more according to their computational performance and the computational performance is the sum of all computational performance.

Before leaving this section, closer look at the individual environments is taken, starting from environment A. In this environment the upper bound increased linearly from 1 minute to about 40 minutes. For example at about the 20 minute mark the upper bound and the simulation models are at about a speed-up of 3. From the view point of the bottleneck model three Bluetooth phones (one master and two slaves) would be sufficient to provide maximum speed-up to the 20 minute problem. This is although not the case for the simulation models (or the basic model), since in the simulations more devices results in

better utilization of the network (dissected in more detail on the next chapter). At about 25 minutes the simulation models start to increase more slowly than the upper bound model. At 40 minutes the upper bound reaches its maximum speed-up of five and from thereon the other models catch up.

In environment B the speed-up rises much more sharply than in environment A and the upper bound reaches a maximum level of 10.3 already at little above the 20 minutes mark. The reason for the steeper curve is that the WLAN network has much greater transmission speed than the Bluetooth network. The speed-up in environment B is the greatest of all environments, which is because the PC can as a slave, provide considerable computing power. A little surprising is the fact that the simulation 10 outperforms the simulation 50. This should not be possible and probably happens because of small rounding errors downward at each discretion step in MATLAB code, and thus, as there are more packages, rounding errors may sum up.

In environment C the upper bound reaches its limit of 1.7 at about 20 minutes. The slopes of the models are much flatter than in the previous environments. Although using WLAN technology the descriptions sizes sent over the WLAN are much larger, hence, the flatter slope. The simulation 10 model deviates clearly from the other models and does not give any speed up. This is because the packages in simulation 10 are too large to be sent to any slave. In other words, the computation power of the master is nearly ten times the computational power of the slaves. Thus, computing on sub-problem on a slave takes about as long as it takes to compute ten sub-problems on the master. If one sub-problem is sent to some slave the time taken to complete that problem is the bottleneck. Also remark that the relative speed-up is much lower in environment C than in any other environment.

Finally, in environment D the upper bound reaches its maximum of 2.5 at 40 minutes. What is special in the environment D is that the focus from crowded network to crowded computational units happens at different minute marks for various sub networks or transmission channels. As the simulation supports only one transmission channel environments, no results of simulations can be presented in this environment.

4.3.2 Varying number of slaves

In this part we present results from fixing the problem and varying the environment. Only environments A, B and C were studied. The problem descriptions were fixed at 20 MB, 60 MB and 600 MB in environments A, B and C respectively. The required computational time on the master device was fixed at about 40 minutes, i.e. $T_{sequential} = 40\ minutes$. The complexity coefficient k and the computational requirement Y were derived in each environment from the computational power of the master device, the description size and the duration of solving the problem on the master device solely ($T_{sequential}$). Moving from environment A to B, both the description and computational power of the master approximately tripled, thus the complexity coefficient was nearly the same both in environment A and B. The same applies when going from environment B to C, where the description size and the computational power of the master in environment C are about ten times those in environment B. Thus the complexity coefficient stayed nearly constant from environment to environment.

In each environment the varying component was the number of slaves. In environment A the number of slaves was increased from 1 to 7, in environment B the number of slaves was increased from 1 to 7, and in environment C the number of devices was increased from 4 to 9. *Table 7* summarizes the environments and problems.

Table 7. The environments and computational problems.

	Environment A	Environment B	Environment C
Master device	Bluetooth mobile phone (104 MHz)	WLAN mobile phone (300 MHz)	Desktop PC (2,8 GHz)
Slave devices	1-7 Bluetooth mobile phones (104 MHz)	1-7 Desktop PC (2,8 GHz)	4-9 WLAN mobile phones (300 MHz)
Communication channel(s)	Bluetooth (560 kbps)	WLAN (6,5 Mbps)	WLAN (6,5 Mbps)
Size (X) [MB]	20	60	600
Sequential solving time (Tsequential) [min]	40,5	40,5	40,5
Complexity coefficient (k) [Hz*s/B * 10^3]	12	11,6	10,8
Computational requirement (Y) [Hz*s*10^9]	253	729	6800

The results are presented in *Figure 8*. In all environments the models start at some initial level and increase to some upper bound as the number of devices is increased. Looking first at the upper bound model, having 2, 2 and 5 devices in environments A, B and C respectively, the upper bound starts at a speed up of about 2.0, 10 and 1.4 in environments A, B and C respectively. In the beginning the bottlenecks are the computational powers of the slave devices, and as the sum of the computational power of the slave devices increases as the number of slave devices increases, the speed-up increases linearly with the number of slaves. The upper bound reaches a maximum plateau level of 5.0 and 16 at 5 and 3 devices in environment A respectively B, and at some later amount of devices in environment C. The restricting factor at the plateau level is the network. Hence, the logic is in a sense reversed to the logic in the previous part where the computational requirement increased and the plateau level was restricted by the computational power of the slaves. At the plateau level, adding devices does not improve speed-up since the network through-put is lower than that of the computational power of the slave devices together.

Looking at the other models besides the upper bound model, they all, except for the simulation 50 model in environment B, are below that of the upper bound. The simulations and the basic model all should always be below the upper bound level, and the exception in environment B was probably because the problem is not of exactly the same computational requirement for the upper bound and the other models. The difference is at the third significant number. The reason why the other models are below (or should be below) the upper bound is that there existed moments during the runs when either the network was idle or the computing devices were idle in both the basic model and the simulation models runs. Again at least one simulation outperforms the basic model. The reason is the same as in the previous chapter, in the simulation models there is in general less idle time during the run.

Further examinations look at individual environments starting from environment A. In environment A the upper bound reaches its turnaround point at 5 devices. Simulation 50 reaches the same level two devices later. Apparently the network can not be fully utilized at 5 devices in the simulation 50 but at 7 devices the network is kept busy the whole time, i.e. the network queue is full throughout the simulation. In the simulation 10 speed-up still rises as the last device is added but does not reach as high as the simulation 50 model. The

basic model is well below the other models but it should reach the upper bound as the amount of devices reaches infinity.

In environment B the upper bound, the simulation 10 and the simulation 50 all reach their maximum level already at three devices, i.e. two slave desktop PCs. The incremental increases are so large when adding PCs, thus, the simulations and upper bound all reach the same level quite quickly and there is no big difference between the upper bound and the simulations. Only the basic model lags behind. The increase in speed-up is largest in this environment as the master device has quite low performance but the network speed is relatively high (WLAN).

In the last environment C the saturation level of the network is not reached with 10 devices. The simulation 10 model is again inefficient as there is no point in providing sub-problems to the other devices.

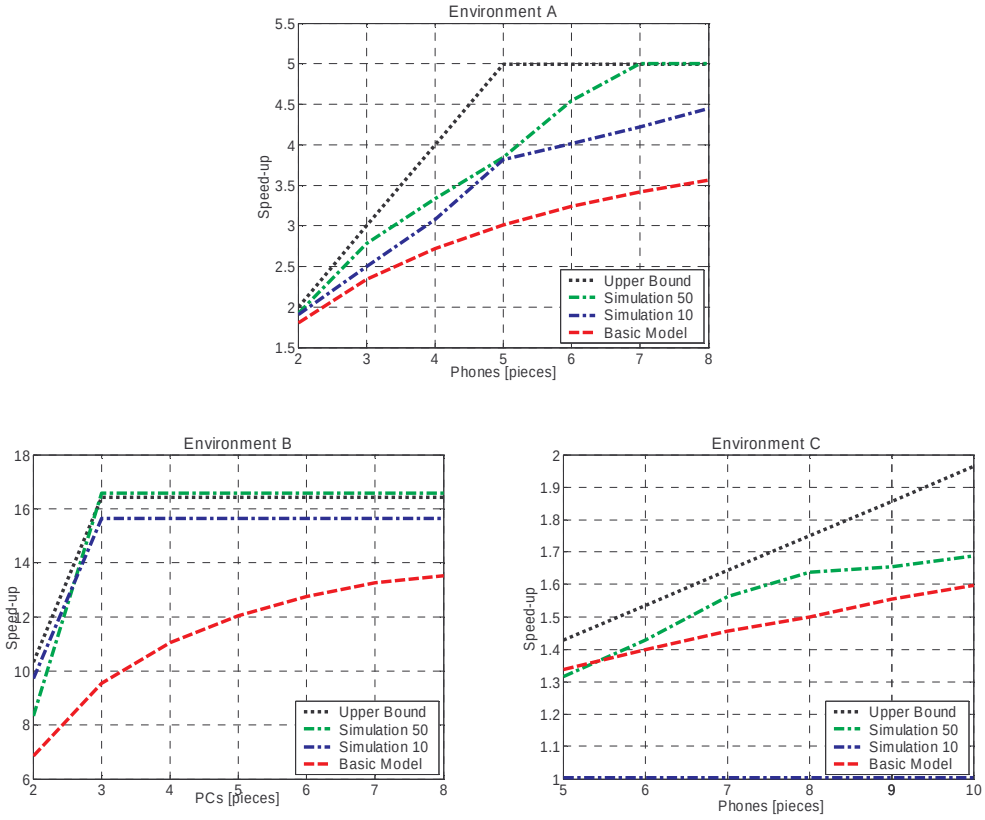

Figure 8. Relative speed-ups as a function of devices in the network in the environments A, B and C. The environments and problems are summarized in *Table 7*.

The notion that speed-ups increase relative linearly to the number of devices but reaches some saturation level was examined also by Jongho Nang and Junwha Kim (1997). Their

environment consisted of a desktop PC with a Superscalar SPARC 60MHz processor as master device and several desktop PCs with MicroSPARC 50 MHz processors as slaves, all connected in an Ethernet. In their setting the master device did not provide any computational power. The problem was to encode a video clip into MPEG-1 format. *Figure 9* shows some of their results. The different trajectories correspond to different distribution schemes (different amount of packages). All distribution schemes resulted in an almost linear speed-up when 1-18 devices were added. When more than 18 devices were added the speed-up increase was limited resulting from the increased communication overhead.

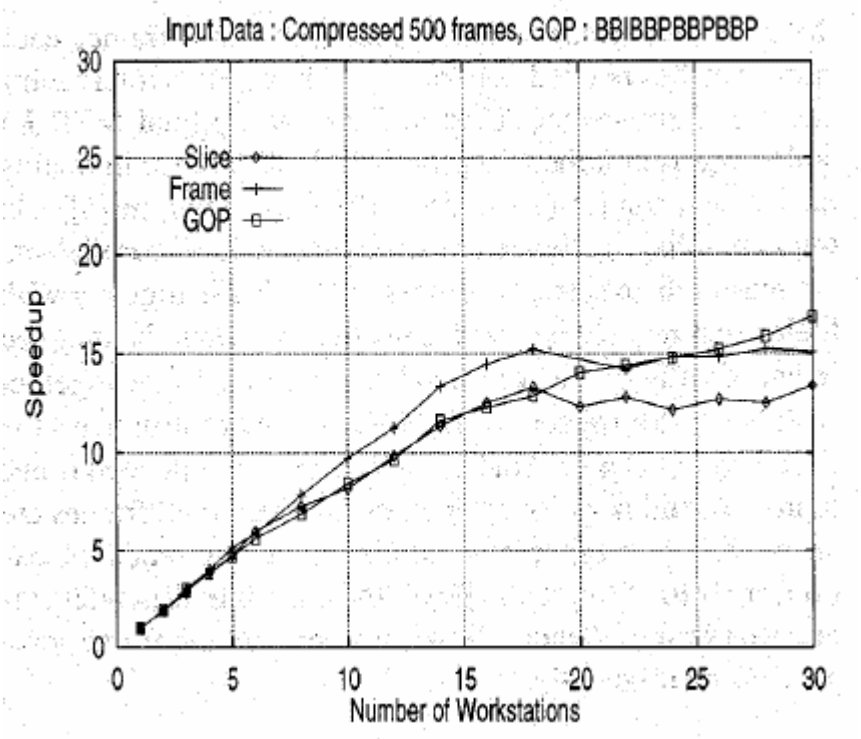

Figure 9. Relative speed-up as a function of number of devices [Nang et al. 1997]

4.3.3 Varying computational requirement and varying number of slaves

In the previous two chapters results from studying speed-ups when the computational requirement of the problem is varying and when the number of slaves was varied were showed. In both chapters one of the parameters was varied while the other one was kept constant. Varying both parameters resulted in surfaces, where on the x-axis the number of slaves varied, on the y-axis the computational requirement of the problem varied, and the z-axis shows the relative speed-up that resulted was plotted. These results are presented in

the appendix C. The results depicted above are just cross sections of the figures in appendix C.

5 Conclusions

Initial assumption about distributing the data as proportions of relative computing powers of devices seems to be correct one. Testing the models did not gave any implications of other type of behavior as long as computational tasks were lot more time consuming than data transfer period, which is the case in signal processing, for example audio and video packing and conversion. This is partly surprising, as queuing effects were assumed to have greater impact than they did.

One interesting finding is that in environment A adding more than 5 mobile phones does not have any affect to the performance. Thus Bluetooth technology as a standard, allowing no more than one master and seven slaves in piconet, does not seem to be restricting component with current computing powers available in mobile phones.

However, both models were relatively heavy simplifications of reality as has been discussed in chapters before. Therefore it can not be stated that computing powers and bandwidths would be the only relevant factors affecting the decision of how to distribute the tasks. But more importantly, models gave clear evidence that computational performance can be increased due to distributed computing.

Speed-up comparison was done in four home environments A, B, C and D. Environment A consisted of phones connected in a Bluetooth piconet. Environment B consisted of a phone to which PC:s provide computational resources over a WLAN channel. Environment C consisted of a PC and several mobile phones that provided computational resources over WLAN. The last environment D consisted of various devices providing computational resources to a PC.

In a setting with a fixed problem and fixed environments, the speed-ups in environment A were 2.7, in B 2.9 and in C 1.5. The examination in D of how the task is distributed by using the basic model scheme for distribution revealed that in home environments the bulk of computational power seems to lie in personal computers and gaming consoles. Thus, these are the essential devices in performing computation for a PC. Furthermore, devices such as handhelds, mobile phones, DVB boxes and other alike that posses limited

computational performance in comparison to PCs or Gaming consoles, can provide only marginal utility to the PC. This was also seen in environment C where the speed up was lowest of all environments. Vice versa, the PC could provide considerable computational gains for their weaker counterparts as was seen in environment B, where the speed-up was largest. In the middle were speed-ups in environment A which were moderate. In this environment similar devices interacted.

As a conclusion, it can be said that the achievable relative speed-ups depend on various issues. The issues studied here were the environment, computational requirement of the problem, the number of slaves in the environment and the distribution scheme. First of all, the environment defines how much computational power is available in the slaves and what kind of communication links are in use.

Second factor is the problem type relative to environment: In problems with large computational requirement in contrast to the description size, the computational power of the slaves could be more fully utilized as the network does not restrict performance as much as in less computationally demanding tasks relative to the description size. Thus, relative speed-up increases in environments as computational requirement increases. But, it is restricted by the processing power of the slaves. In other words, as the computational requirement increases to infinity the speed-up converges to the sum of all computing devices in the network divided by the computational power of the master. Fixing the environment as in *Table 6*, this upper bound in environment A was 5, in B 10.5, in C 1.7 and in D 2.5.

Third, increasing the number of slaves in the network increases the total amount of computational power in slaves, hence, increasing speed-up, as long as the network bandwidth does not restrict increases. Fixing the problem as in *Table 7* showed that increasing the number of slaves increased speed-up nearly linearly in the beginning, but as the number of slaves was increased sufficiently the network started to get saturated and the speed-up converged to an upper-bound. The upper bound in the environment A was 5.0 and 16 in the environment B. Showing that the WLAN technology provided a higher speed-up saturation point, because of its higher bandwidth (although the master had higher computational power in B). The upper-bound was reached at different number of slaves for different distribution schemes. The simulation 50 converged fastest to the upper bound

and approximately reached it at 6 slave phones in environment A and 2 slave PCs in environment B. The faster converging in environment B was due to the higher computational performance of the slaves. In environment C the upper bound was not reached with 9 slave devices yet.

The last factor to bee mentioned was the distribution scheme. If the problem was divided into smaller pieces (simulation 50), or if the problem is such that the computation on the slave may start when the problem starts to arrive at the slave (upper bound) it may increase the speed-up as leading to better utilization of the environment resources. This could be seen in the upper-bound and the simulation 50 always outperforming the other distribution schemes. This was because these schemes could more fully exploit the resources in the environment, and thus, having less idle time during the runs.

Summarizing, the results are intuitively quite clear, the best situation in terms of speed-up is when we have a lot of surrounding computational power, sufficiently fast networks, computationally demanding problems relative to the description size, and we can utilize resources efficiently. The speed-up then depends on (among others) how well these factors are met.

5.1 Study's focus and the real world

The results presented in this study are only guiding, since there are strong assumptions and simplifications underlying them. In real life perfectly decomposable problems are rare and there might exist multiple factors that delimit the efficiency of distributed computing. To name a few: In this study no fixed costs existed. Thus, it was always lucrative to distribute the problem. However in real life there might be considerable costs in composing and decomposing the problem, setting up communication to slaves (e.g. setting-up a piconet in Bluetooth environments), and in the need to transfer software components in advance of any computation in slaves. Moreover, the distribution scheme might need to send control information during the run, and thus communication link properties like latencies might start to play considerable roles. This is particularly a weakness of our simulation procedures, in which it was assumed that the master has perfect information of e.g. computing queues of slaves. In addition, a fraction of computational power will be lost due to the software controlling the distribution. Also other issues might need to be included,

like uncertainties of performance and the fact that different computers have different hardware and system properties which significantly impact how well the a task of one device can be solved on another. One reason why none of the above mentioned issues were dealt with in this study is that their impact emerges from specific applications and technologies.

It is though worth to noticing that with a few modifications our results can include some of the issues mentioned above. As an example, consider a case where there does exist a constant decomposition and composition cost, and decomposition and composition must be done separately from other procedures. Assume the decomposition and composition cost is all in all half of the time taken to solve the problem sequentially on the master solely. This implies that the break even speed-up is 2, i.e. the distribution is worthwhile only at speed-up above 2.

5.2 How to utilize the speed-ups

Since speed-ups are available, some questions naturally emerge. For example, what kind of applications are and when are they attractive for distributed computing becomes an important topic. From the view point of this study, applications that have properties similar to the previously mentioned factors that enhance speed-up are most lucrative and this study can provide some guidance into what kind of speed-up can be achieved. Still, no absolute answers can be given since many issues affecting distribution emerge from the application itself. First, the problem might be naturally situated on a particular device. For example, a device (the problem owner) needing conversion of a small video-clip might be a cellular phone with video viewing possibilities. Whether the problem might need distributed computing or not in the first place depends on if it needs any speed-up at all and what kind of speed-up. There might exist some threshold speed-up that in the eyes of the application user is desirable or even necessary, but above this the utility to the user is not so considerable. For example, a video-game that has an artificial intelligence might not be appealing to play when responses from the computer take three minutes, but if one minute responses can be achieved, i.e. relative speed-up of 3, the situation might be totally different and the game might become interesting to play. The next issue is of course what kind of speed-up the problem and the technologies are capable of. If the problem owner is a low performance device it might gain considerable speed-up from a more high

performance device. Still, it has to be considered what kind of devices can be assumed to exist in the same environment as the problem owner. For example assuming that a mobile phone has on average access to a piconet of 4 similar other devices, the results in this study tells us what kind of possible speed-up we might achieve. The final speed-up then depends at least on how large the computational requirement of the problem is relative to the description size and how the application can be distributed on an algorithmic level. If, for example, we have a mobile phone with the video conversion problem mentioned above, we would require a speed-up of 2 in order to be able to view the video concurrently during the conversion process. If there it is assumed that the phone is on average surrounded by four similar devices, the video stream is of size 20 MB and would require conversion of 20 minutes, the problem is nearly perfectly decomposable, and can easily be decomposed to 10 smaller video clips (e.g. in one minute), then our results suggests that the threshold level seems to be achievable and a speed-up of 2,5 is possible. Thus, further examination would be desirable.

There are several domains in which distributable problems can be found. For example, applications in signal processing, video and audio format conversions, and image processing can be demanding. The signal can in many cases be split into smaller slices, thus making it decomposable. As an example, MPEG coding schemes were presented by Nang et al. (1997). Also, pattern recognition and voice recognition might have their appeals. In voice recognition the front-end device recognizes characteristics of the voice. These characteristics could be distributed for further processing. Delaney and Simunic (2003) present a scheme for distributing voice recognition of mobile phones. Moreover, the gaming domain is one that could also be source of applications. For example, artificial intelligence might be a computationally demanding problem. Here for example the problem of distribution probably is how a game tree should be traversed. Also, modeling of physical movement and other concerns in real time games can be demanding. Besides the above mentioned there probably exists a wealth of domains in which suitable problems emerge, e.g. cryptography, mathematical (optimization) algorithms and so on.

Although this study was only concerned with computational speed-up, there are other types of gains that can be achieved. In addition to computational power other devices may contribute with storage and memory. Additionally other concerns might give the incentive to distribute the problem. In Delaney's and Simunic's (2003) voice recognition example the

primary reason mentioned for distributing the speech recognition done by a mobile device is that it minimizes the battery consumption of the mobile device. Although dissimilarities of devices is often a problem for distributed computing, dissimilarities can also lead to interesting settings where one device has a certain weakness and utilizes another devices strong ness on the same point.

In the future it is reasonable to assume that computational power in homes does increase. Already, Moore's law does give reasons to believe so. But, this does not guarantee much more interesting settings for distributed computing. Other trends, however, that might improve the benefits of distributed computing are converging hardware and systems (a move towards this is the Sony patent application by Suzuoki et al., 2002) and higher connectivity of devices in home environments. Furthermore, it is possible that smaller devices like mobile phones, handhelds and other devices alike, which are connected and can contribute to computation, may increase in number in future homes. To take a somewhat futuristic view, in addition to the existing handhelds, in the future there might exist robots, like the Sony Aibo dog, smart refrigerators and other home appliances that might possess computational power that can be exploited. Hence, the relative computational significance of weaker than PC devices might increase as they increase in number. Similarly, Phan et al. (2002) denote that the sheer number of wireless mobile devices is a reason per se why these devices are a great untapped resource, though their consideration is in large grids like SETI@home.

6 Proposals for further studies

This chapter discusses the aspect of pertinent further research as well as the possible future developments on the field of distributed computing. A variety of, for some qualities, extreme cases of modeling the problem is presented thus giving an overview of the possibilities to develop or reformulate the modeling further.

6.1 Fixed-sized packages

If packages are required to be fixed-sized and can not be divided in a desired way, it could be useful to know something about the contents of the packages and processing of them in the system. By this we mean that when deciding about the allocation of the fixed-sized packages the algorithm should take into consideration the contents and nature of the ongoing processes and the queuing packages. In real life situations some packages could be connected to each other so that they contain partly same data and the similar parts could be computed faster using the same processing unit or the packages should be handled in specific order or successively. This kind of study is omitted in our paper, but offers conditions for better analyses of some computing problems.

6.2 Complexity

Complexity was by far the most difficult quality of the computing problems to model. This is due to the fact that there is no indicator that would fit to all kinds of problems to determine a comparable measure of complexity. We have modeled the complexity as a "density" of computing operations in an amount of raw data processed. This amount of computing operations is assumed by us to be constant in the data. This is not a very realistic assumption in many problems and thus it would be a good improvement to include a possibility to vary this value in the model.

This improvement would require a specific model for each problem type because the problem types are very different in nature. For example, in a video format conversion or compression problem the complexity could be a function of computing operations related to the pixels on screen and the change in them. In a quiet scene with no movement and

simple colors and shades the operations density would be low. On the other hand, in a scene with a lot of movement it would be drastically higher.

The complexity modeling function for prime number calculations or chess problems, for instance, would have to be completely different because these problems result in a problem tree. This study aims to cover some more or less common cases to explain distributed computing in home environment and devices, not going very deep into different specific computing problems. Knowing what is wanted to compute, it would be an important step to analyze specific problems further as well.

6.3 Simulation approach

The simulation approach could also be exploited in ways other than that was chosen in this study. Describing the problem in detail is the most challenging part of the simulation. The model used for simulation was actually chosen because of the aspiration to model a computing problem with indiscriminate fixed-size packages that cannot be divided and must be calculated as whole. The simulation of distributing computing problems can be studied further and used to test different computing environments. The simulation model used could handle only one type of connections, i.e. one speed, between devices at one time (the speed can be altered, of course). This was mainly because in the end the primary focus was on such environments in which connections were homogenous, and thus the model was built to meet these systems. It would offer more flexibility, if the model could include different connection speeds simultaneously.

6.4 Continuous stream of computational tasks

In this study the computational complexity has been kept fixed in each model. Also, in all models tasks enter the system at the very beginning when the no time has elapsed. Distribution might behave differently, and be much more difficult to model, if the complexity coefficient were different for every sub-problem and the sub-problems would enter the system in various times (i.e. there would exists a continuous non-deterministic stream of problems). This might match the real life situation better since future tasks are not deterministically known in every situation. For instance, user might decide to perform certain new operation while the previous ones still in process.

6.5 Literature Study

We pretty much started our study by looking into some literature in the field of performance modeling and measurement of computational grids, applications and many other fields related to grids and distributed computing. Initially we found no interesting body of knowledge that could be utilized. Looking back the main reason for this was looking in the wrong types of publications and databases (mostly periodical journals). Later finding a wealth of conference papers dealing with interesting subjects we can conclude that there exists a wealth of interesting information on grids and mobile phones connected to grids. As there were not sufficiently time to complete a full literature study in the frames of the course we refer future studies to take a closer look into literature, because fruitful knowledge could be assemble thereof.

7 References

Chia Chong Hooi, Beg M Salim; MPEG-4 Video Transmission over Bluetooth links; 2002 IEEE International Conference

Delaney Brian, Jayant Nikil, Simunic Tajana; A WLAN Scheduling Algorithm to Reduce the Energy Consumption of a Distributed Speech Recognition Front-End; ESTIMedia 2003: First Workshop on Embedded Systems for Real-Time Multimedia

Forum Nokia; Games over Bluetooth: Recommendations to Game Developers; version 1.0 2003

GCIC; Grid Computing Info Centre; URL: http://www.gridcomputing.com [referred April 19th 2004]

IBM [1]; Uni Pentium 4 desktop SFF workstation specification; URL: http://www-132.ibm.com/webapp/wcs/stores/servlet/ProductDisplay?storeId=1&langId=-1&categoryId=&dualCurrId=73&catalogId=-840&partNumber=622020U [Referred April 19th 2004]

IBM [2]; ThinkPad G40 product information; URL: http://www-307.ibm.com/pc/support/site.wss/quickPath.do?quickPathEntry=2388mg1 [Referred April 19th 2004]

Infopen Archivum; 100 Mbps Switched hubok; URL: http://www.infopen.hu/archivum/97/9712/tabla.htm [Referred April 19th 2004]

InfoSyncWorld; Samsung SGH-i700; URL: http://www.infosyncworld.com/hardware/whandhelds/n/20.html [Referred April 29th 2004]

InfoSyncWorld; Nokia 6600; URL: http://www.infosyncworld.com/hardware/smartphones/specs/9.html [Referred April 29th 2004]

Jongho Nang, Junwha Kim; An Effective Parallelizing Scheme of MPEG-1 Video Encoding on Ethernet-Connected Workstations; 1997 Advances in Parallel and Distributed Computing Conference (APDC '97)

Kuang Hairong, Bic Ludomir F., Dillencourt Michael B., Chang Adam C.; PODC: Paradigm-Oriented Distributed Computing; The Seventh IEEE Workshop on Future Trends of Distributed Computing Systems December 20 - 20, 1999

Kumm H. T., Lea R. M.; Parallel Computing Efficiency: Climbing the Learning Curve; TENCON '94 IEEE Region 10's Ninth Annual International Conference; Proceedings of 1994.

Lunheim Trygve, Skavhaug Amund; An Experimental Testbed for Using WLANs in Real-Time Applications; 2002; URL: http://www.hurray.isep.ipp.pt/rtlia2002/full_papers/17_rtlia.pdf [Referred April 19th 2004]

Mak Victor W, Lundstrom Stephen F.; Predicting Performance of Parallel Computations; 1990; IEEE Transactions on parallel and distributed systems Vol. 1. No. 3

Nokia; Nokia Mediamaster S260 Specifications; URL: http://www.nokia.com/nokia/0,8764,27196,00.html [Referred April 19th 2004]

Phan Thomas, Huag Lloyd, Dulan Chris; Challenge: Integrating Mobile Wireless Devices Into the Computational Grid; 2002; URL: http://delivery.acm.org/10.1145/580000/570679/p271-phan.pdf?key1=570679&key2=9424532801&coll=GUIDE&dl=ACM&CFID=20413831&CFTOKEN=19907503 [Referred April 19th 2004]

SETI@Home; SETI@Home project's home page; URL: http://setiathome.ssl.berkeley.edu [Referred April 19th 2004]

Sprague Luke, Boxall Robert; Of wired and wireless; Network Magazine Vol. 17 Iss. 4 pg 34.

Suzuoki Masakau et al.;Computer architecture and software cells for broadband networks; 2002 United States Patent Application nr 0020138637

Thiébaut D; Parallel Programming in C for the Transputer; 1995; URL: http://cs.smith.edu/~thiebaut/transputer/toc.html [Referred April 13th 2004]

Tom's Hardware Guide [1]; Welcome The Latecomer: Pentium 4 Prescott 3.4 GHz; URL: http://www6.tomshardware.com/cpu/20040322/prescott-06.html [Referred April 19th 2004]

Tom's Hardware Guide [2]; Nokia Phone Meets the Gameboy; URL: http://www6.tomshardware.com/consumer/20031030/nokia-02.html [Referred April 19th 2004]

Urosevic Anna; Bracing for Bluetooth's bite; Wireless Review Vol. 18, Iss. 5 page A14; 2001

Usb.org; USB Frequently asked questions; URL: http://www.usb.org/faq/ans2#q1 [Referred April 13th 2004]

Appendix A: Variables and coefficients

Table 5. The table outlines the variables and constants used in the document. The first column indicates the symbol of the variable, the second column gives a description of the variable and the last one denotes the measurement unit used in this study.

Symbol	Description	Unit
P	A computational problem that is to be solved.	-
p_i	A computational sub problem of P. There are n sub problems p_i for P, and their union is (in some sense) the problem P.	-
Y	The amount of computation needed to solve problem P. In the study Y is measured as the product of the time taken to solve the problem P on a particular device (e.g. on the host device) and the processor performance of this device. That is, if processor performance was measured as calculations per second and the time to solve a problem is in seconds, the units would be calculations. In this study we measured computational performance in MHz, thus the unit of Y is MHz*s.	(M)Hz*s
y_i	The amount of computation needed to solve sub problem p_i. We define perfectly decomposable problems to have the property that the sum of all y_i is Y.	(M)Hz*s
X	The size of a file describing the problem P. In the document X is called description (of problem P) merely. For example, when the problem is to encode a video clip from some original format, X is the size of the original video clip.	(M)Bytes
x_i	The size of a file describing sub problem p_i. In the document, x_i is called description (of problem p_i) merely.	(M)Bytes
R	The size of a file that is the response or answer to the problem P. In the document, R is called answer or response (of problem P) merely. For example, when the problem is to encoding a video clip from some original format, R is the size of the encoded video clip.	(M)Bytes
r_i	The size of a file being the response to the sub problem p_i. In the document, r_i is called answer or response (of sub problem pi) merely.	(M)Bytes
β	The proportionality constant describing the size of the answer relative to the description. When β is equal to 1, a sub problem requires as large an answer as description.	none
k	A complexity coefficient of a sub problem p_i, which is the ratio of y_i to x_i	Hz*s/B
n	The number of sub problems P is divided into.	pieces
m	The number of computing devices is $m + 1$. The index 0 stands for the master or host device. The indices 1,…,m are the surrounding devices providing computational power to the host device.	pieces
c_i	The processing performance of the i^{th} processing unit. There are $m + 1$ processing units.	(M)Hz
s_i	The bandwidth of the communication link from the host to the i^{th}	(M)bps

	device	
$s_{technology}$	The maximum bandwidth of some communication link technology, e.g. s_{BT} is the maximum capacity of a Bluetooth channel.	(M)bps
l_i	The length of the communication link. The time taken to send a minimum size package over a communication link (i.e. latency).	(m)seconds
α	A constant in the interval [0,1] that indicates the proportion of processor capacity used while transmitting data over some communication link.	none
$T_{decompose}$	The time taken to decompose a problem P into n sub problems	seconds
T_{solve}	The time taken to solve n sub problems	seconds
$T_{compose}$	The time taken to compose an answer to a problem P from n sub problem answers	seconds
$T_{distributed}$	The time taken to solve a problem P by distributing it. $T_{distributed}$ is based on $T_{decompose}$, T_{solve} and $T_{compose}$.	seconds
$T_{sequential}$	The time taken to solve a problem P on the host device only.	seconds
t_i	The time taken to solve sub problem p_i. t_i is based on $t_{i,send}$, $t_{i,solve}$ and $t_{i,receive}$.	seconds
$t_{i,send}$	The time taken to send the description x_i to the device, which has been allocated to solve the sub problem p_i.	seconds
$t_{i,solve}$	The time taken to solve the sub problem p_i on the device allocated for this task.	seconds
$t_{i,receive}$	The time taken to receive the answer r_i from the device, which completed sub problem p_i.	seconds
G	The gain or relative speed up resulting from distributing a problem. This is measured as the ratio of $T_{sequential}$ and $T_{distributed}$. Also the ratio $T_{sequential}$ and T_{solve} is used.	none

Appendix B: Simulation results

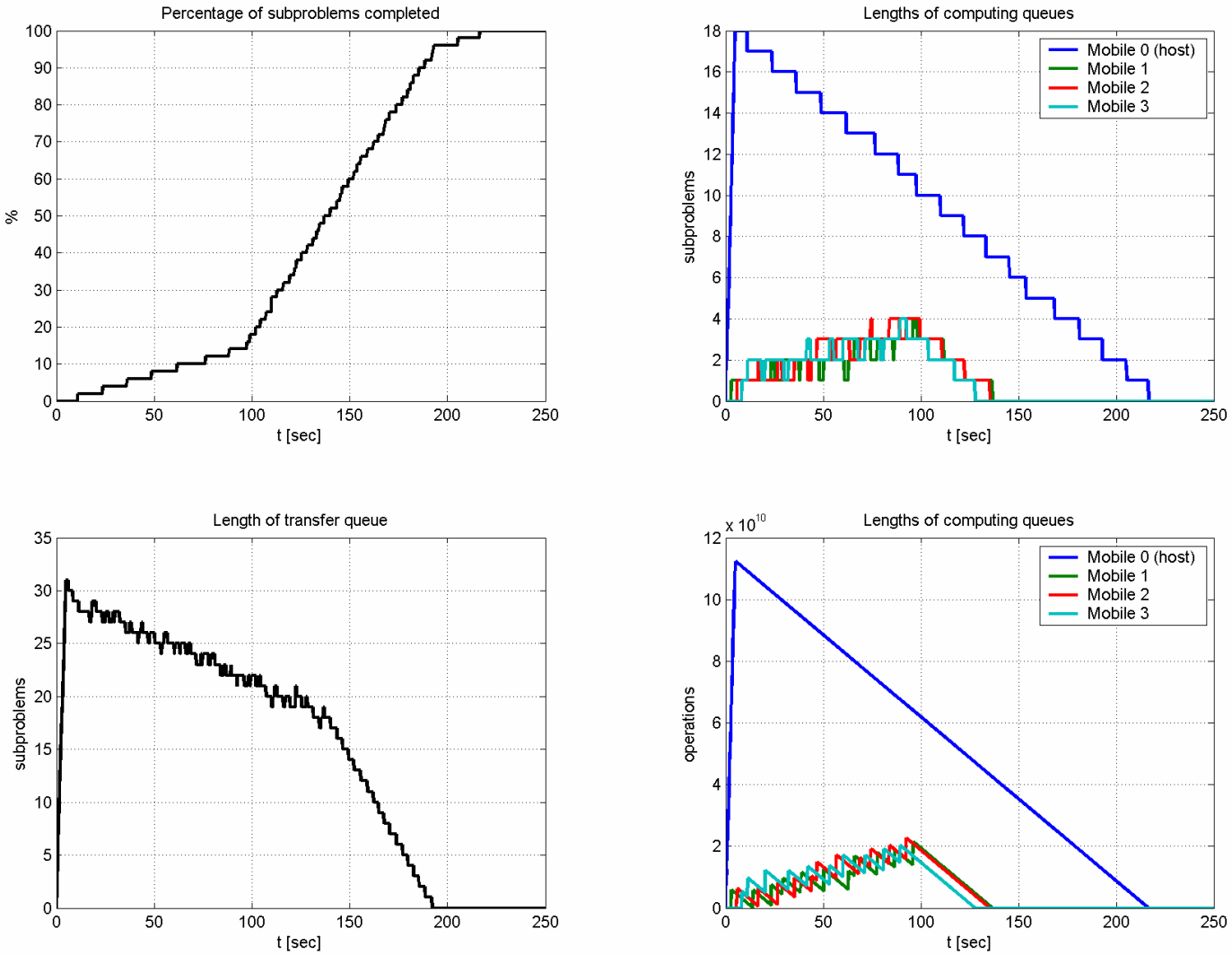

Figure 10. Environment A with 10MByte packet.

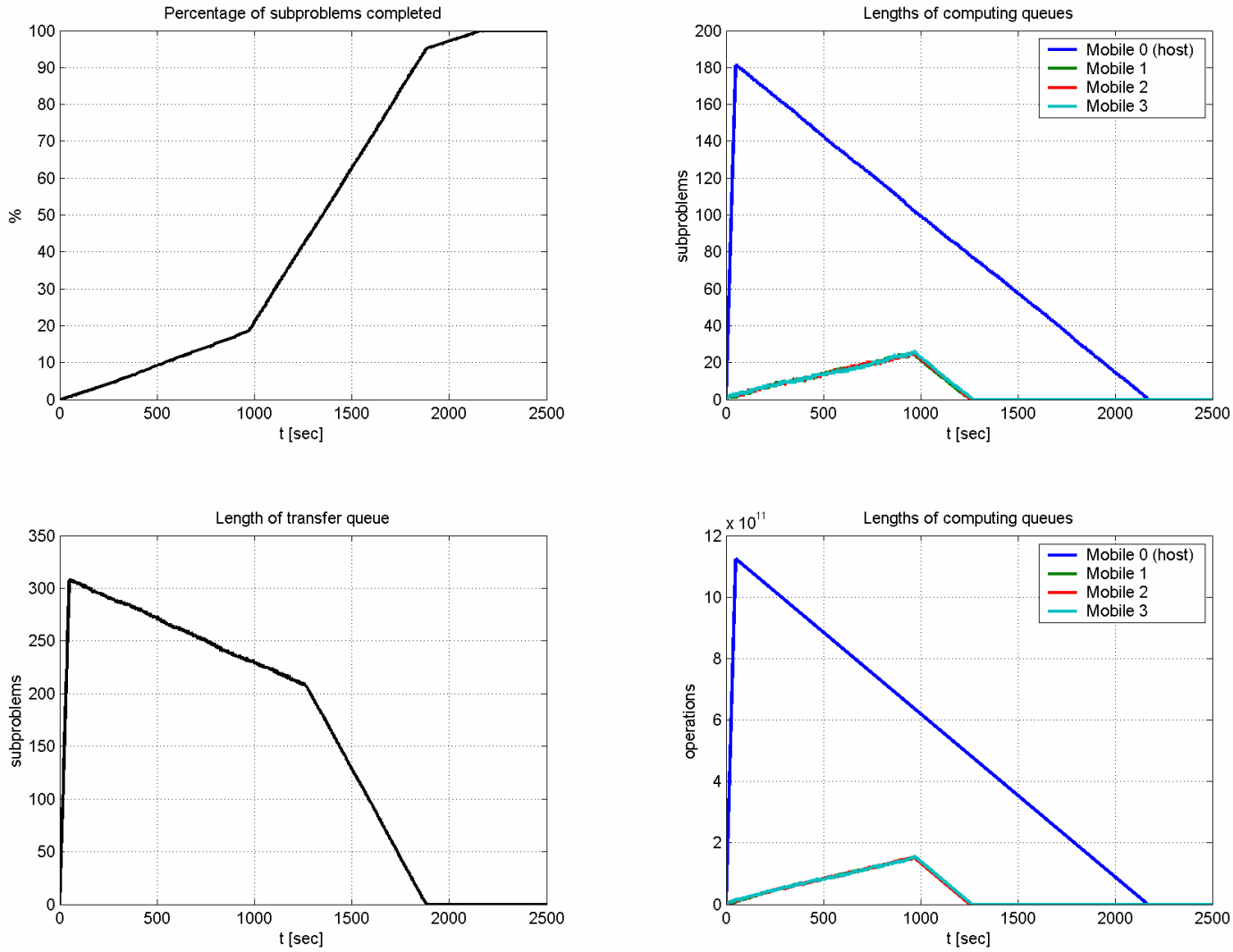

Figure 11. Environment A with 100MByte packet.

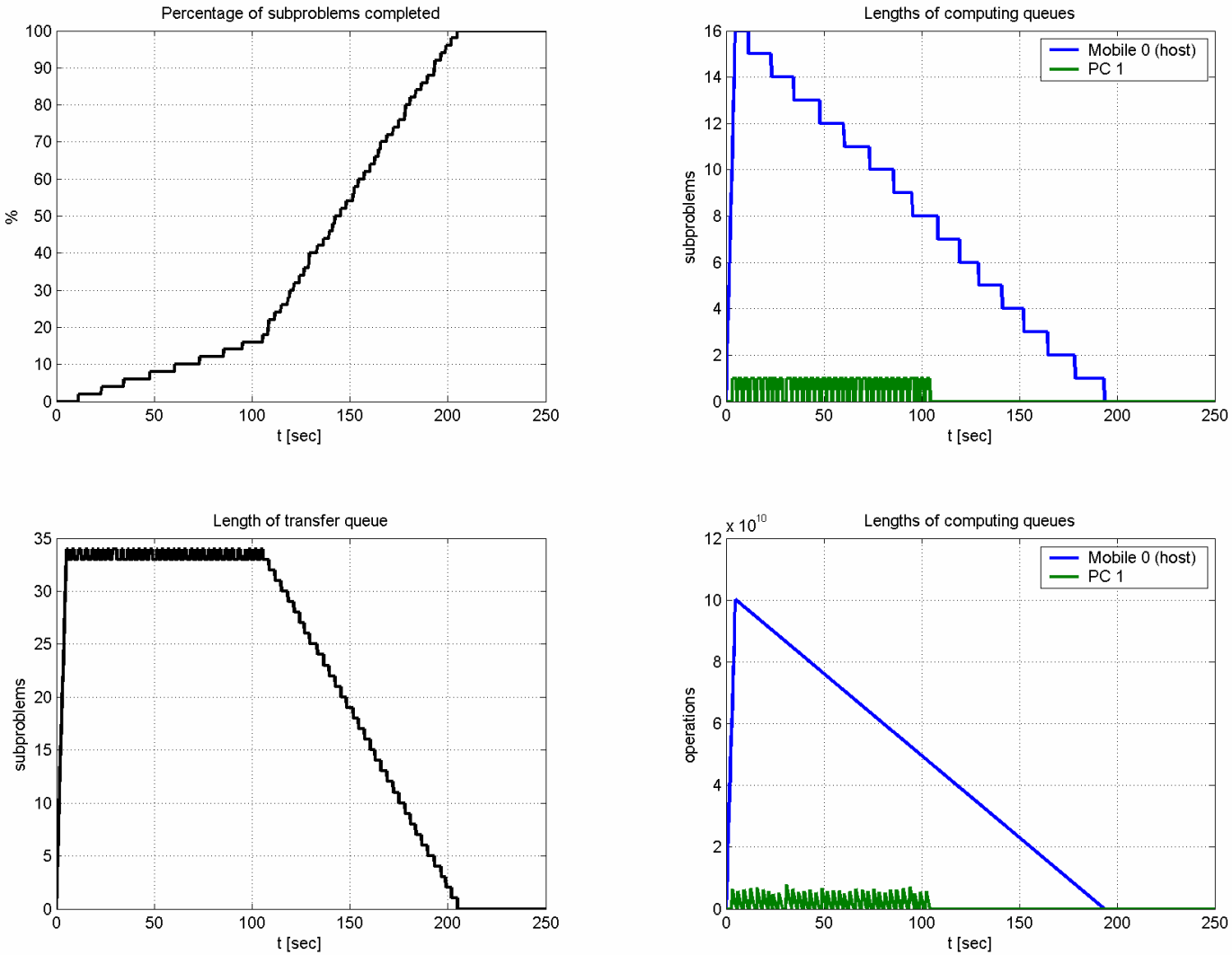

Figure 12. Environment B with 10MByte packet.

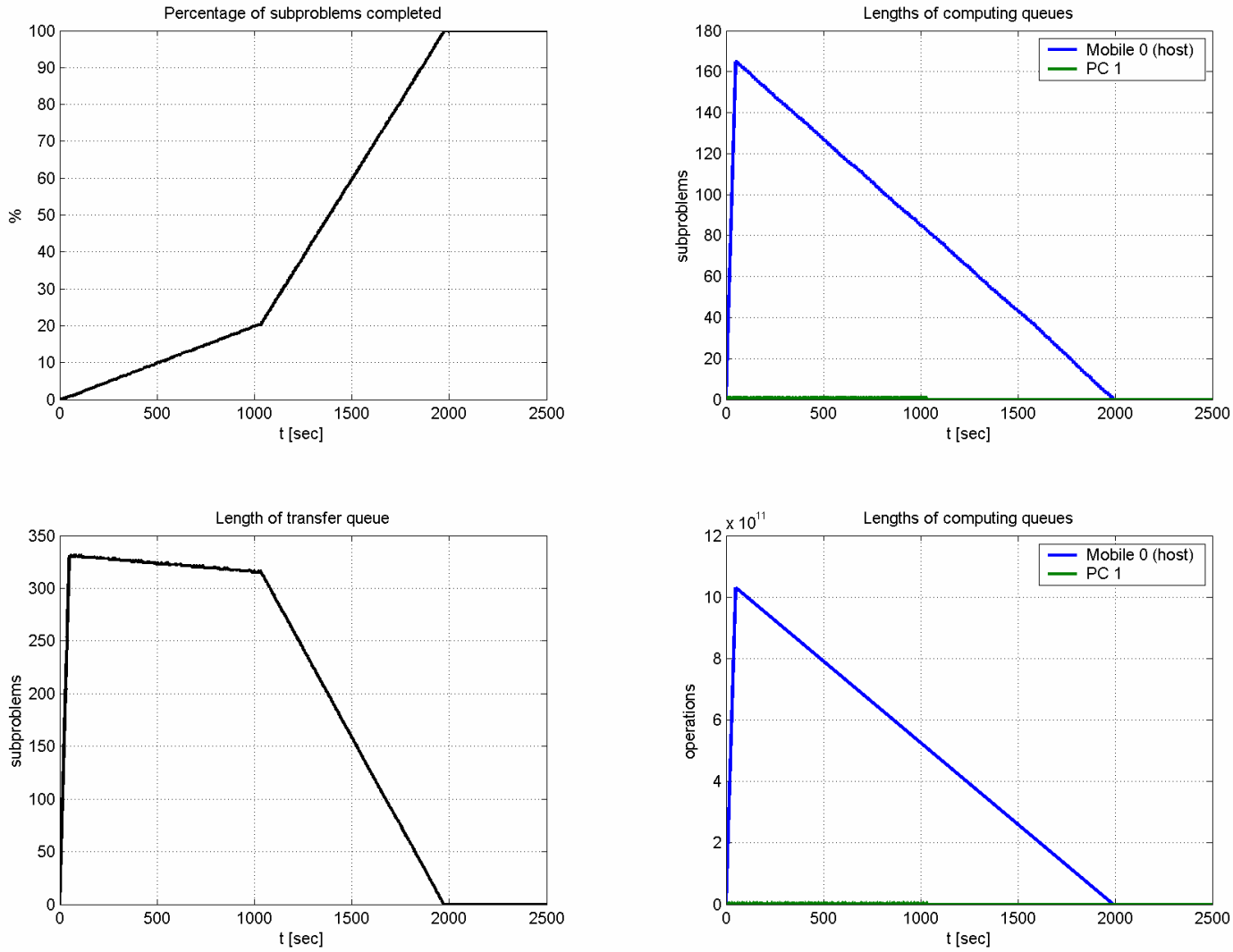

Figure 13. Environment B with 100MByte packet.

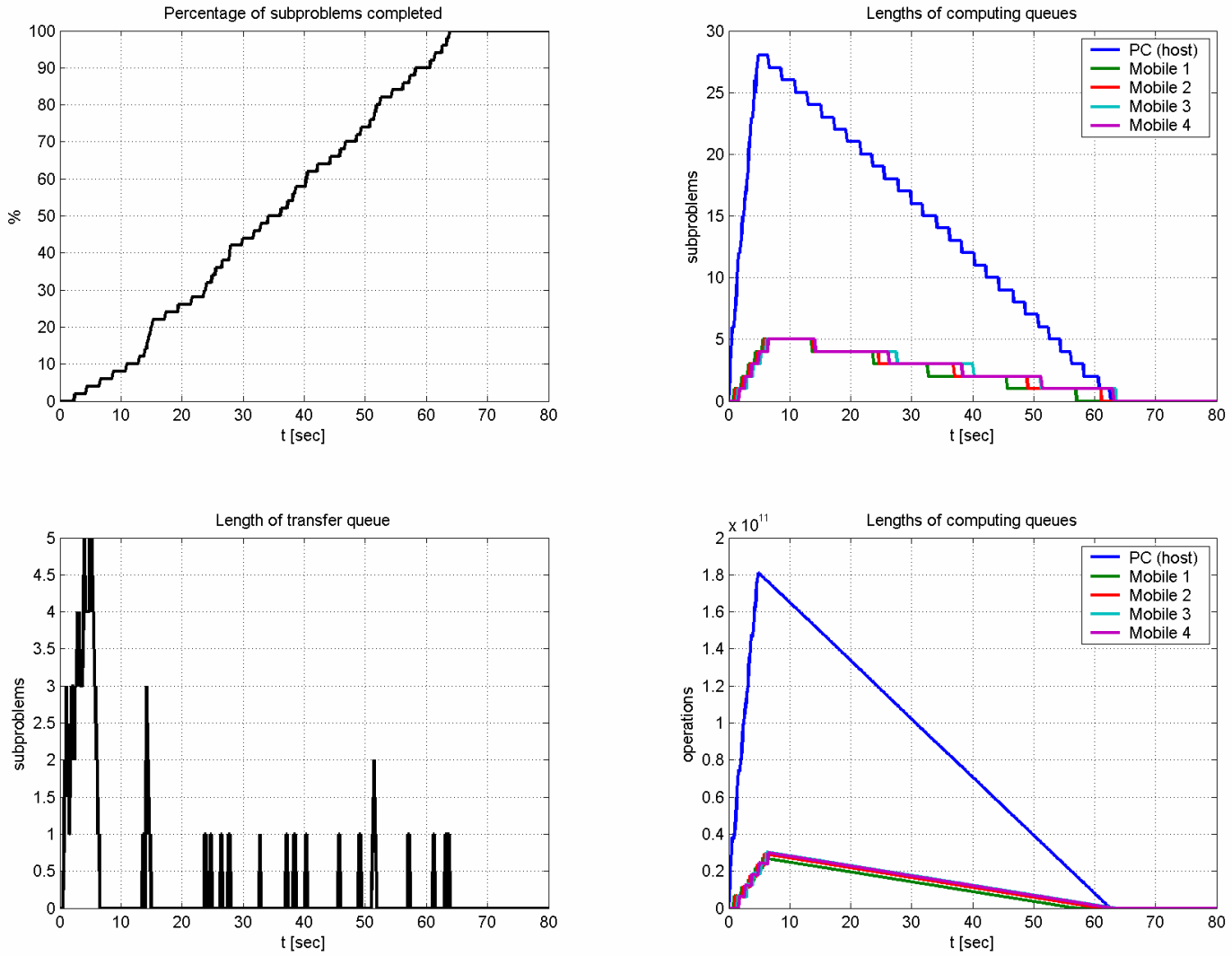

Figure 14. Environment C with 10MByte packet.

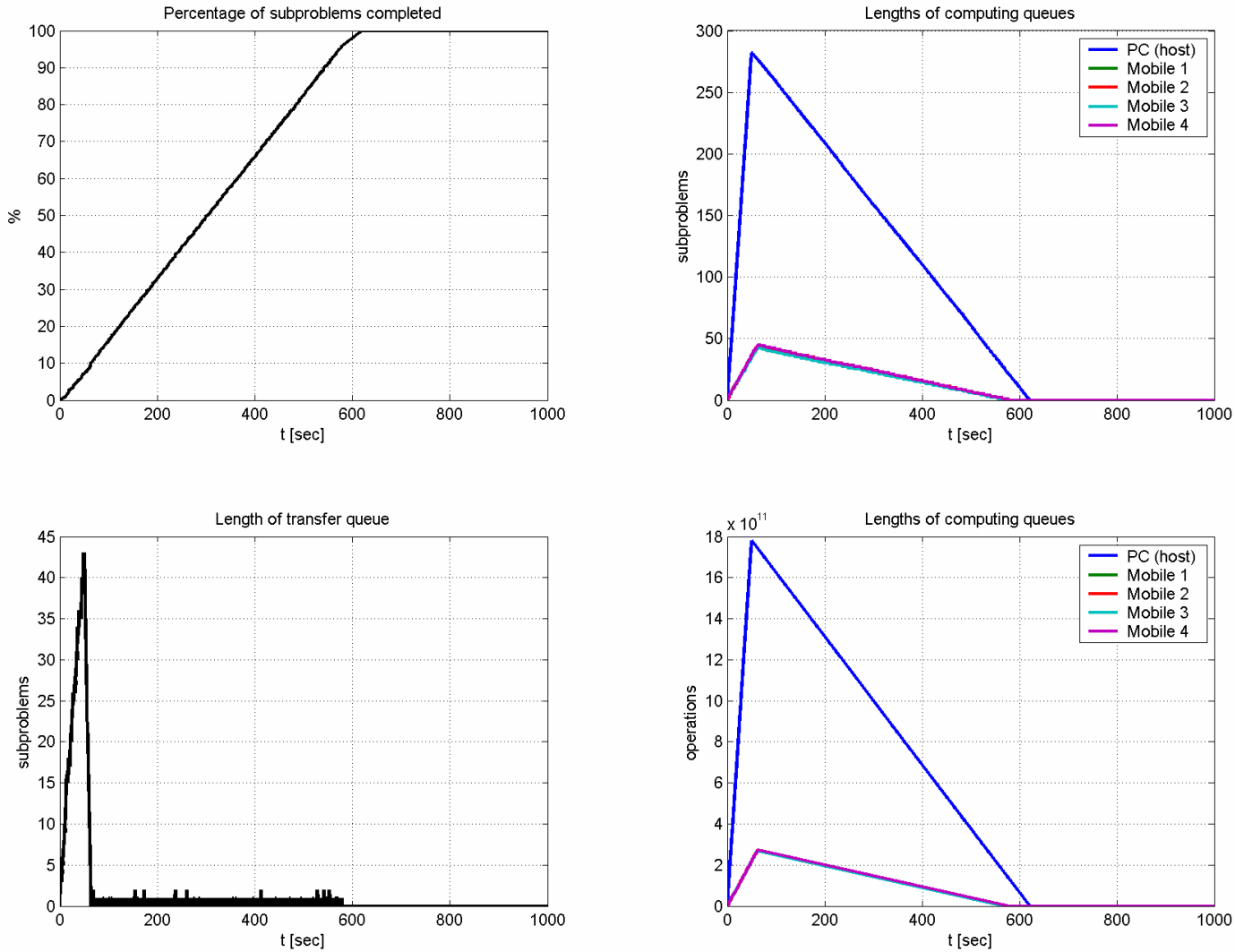

Figure 15. Environment C with 100MByte packet.

Appendix C: Model Comparison

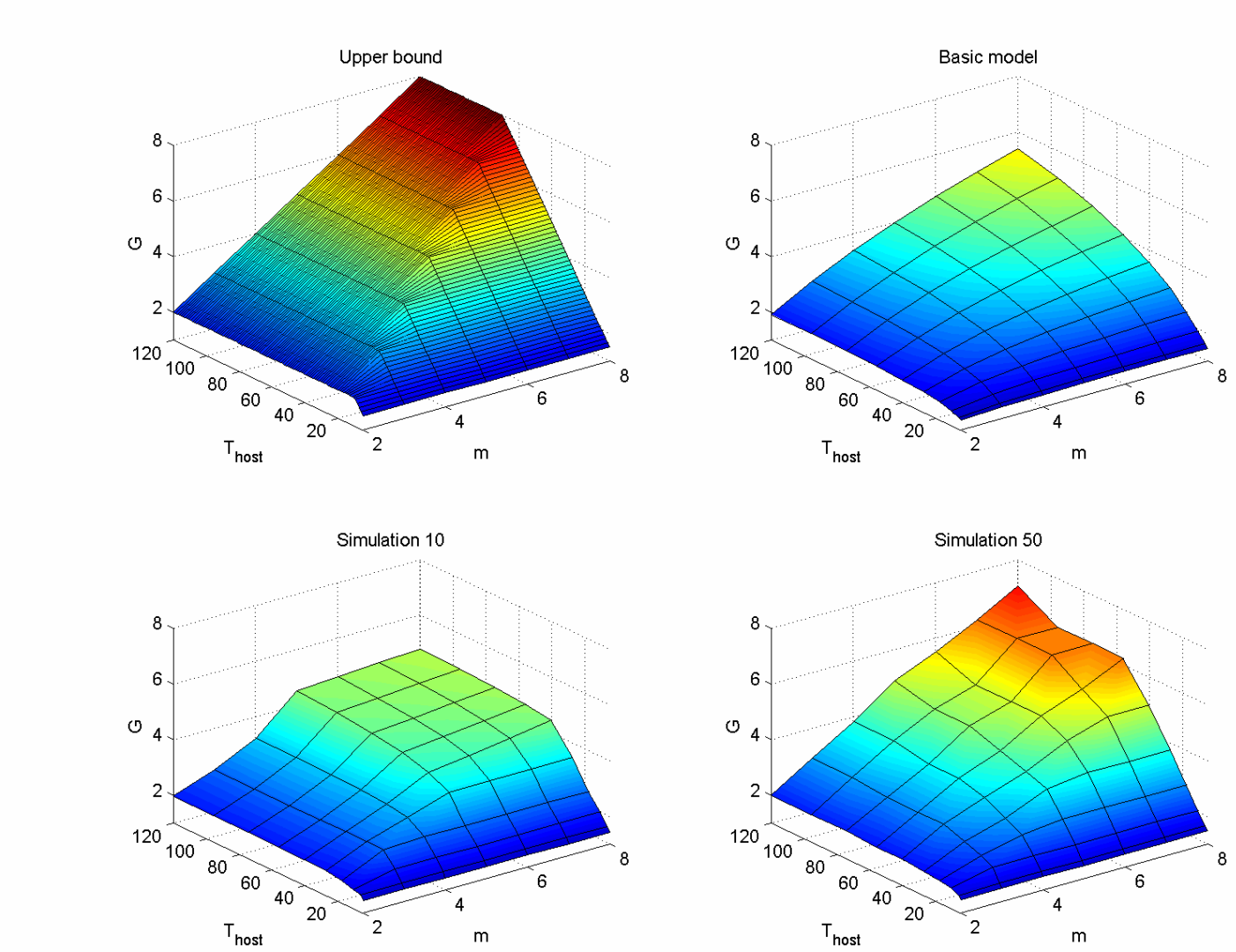

Figure 16. Environment A.

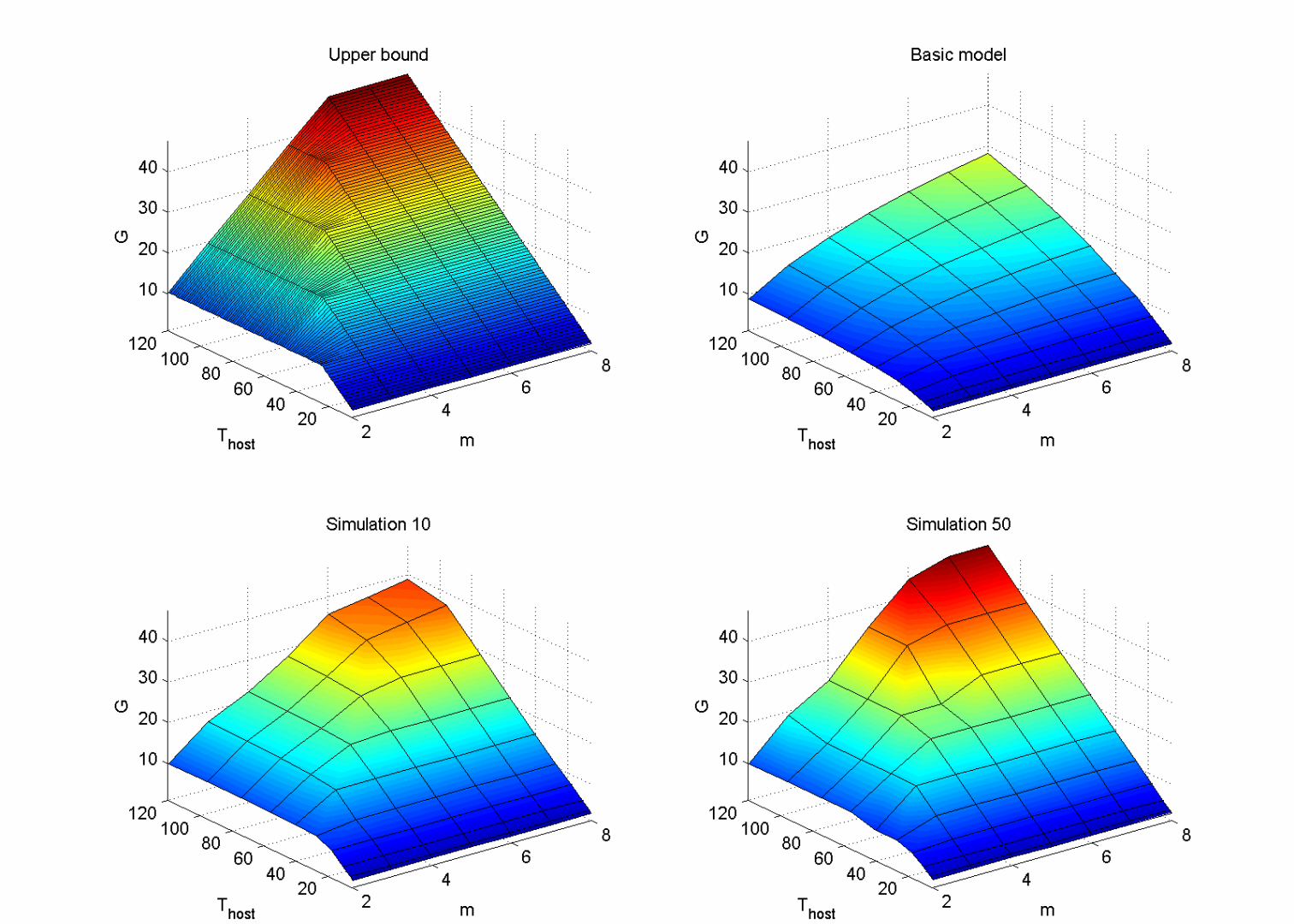

Figure 17. Environment B.

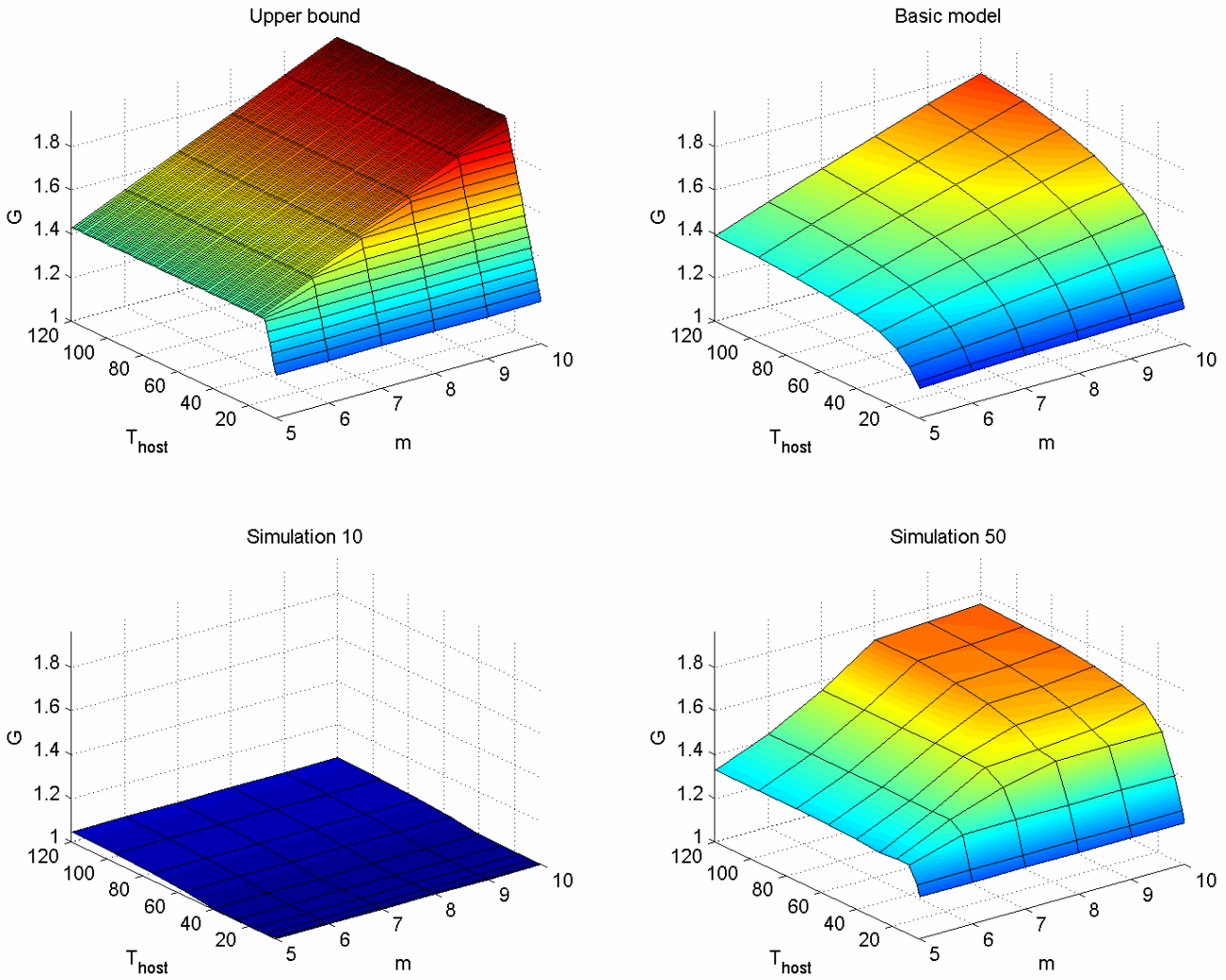

Figure 18. Environment C.

Appendix D: MATLAB code for simulation model

simulation.m

```
% ------------------------------------------------------------------------
%  SIMULATION PROGRAM FOR DISTRIBUTED COMPUTING
%  18.04.2004
%  authors:  Danielsen, Hytönen, Kuusela, Ylä-anttila
%
% This program simulates distributed computing with bag-of-tasks type
% problems in the environment of n computing nodes connected to the host
% computer via single communications link. The system receieves subproblems
% that can either computed on the host computer or trasmitted over the
% communications link to other computing nodes in the network.
% ------------------------------------------------------------------------

% Load constants and generate problems used. Problems are structs defined
% in sim_constants.m and they are generated with function
% generate_problems.m
sim_constants;
problems = generate_problems(num_prob);

% generate discrete time steps
t = [t_start: delta_t: t_end];

% Here are the queues where problems are stored when they await transfer
% or computation or are under transfer or computation. Queues are struct
% arrays with tr_queue containing problems awaiting transfer and host_queue
% and node_queue are the computing queues for host computer and other
% nodes, respectively.
tr_queue = [];
host_queue = [];
node_queue = {};
for i=1: num_nodes
    node_queue{1,i} = [];
end

% list for completed problems
ans_list = [];

% Lengths of different queues (in number of problems). Queue lengths are
% vectors with components xxxx_queue_length(i) corresponding to the length
% of queue at time t_start + (i-1) * delta_t.
tr_queue_length = zeros(1, size(t,2));
host_queue_length = zeros(1, size(t,2));
node_queue_length = zeros(num_nodes, size(t,2));

% number of operations left in each computing node
host_queue_calc = zeros(1, size(t,2));
node_queue_calc = zeros(num_nodes, size(t,2));

% percentage of problems completed
subp_comp = zeros(1, size(t,2));

% Main loop is calculated in delta_t time intervals. It first stores the
% current queue lengths and then checks if new problems have arrived to the
% system. New problems are either copied to host queue or transfer queue
% depending on whether it takes a shorter time to compute the problem on
% the host computer than to transmit the problem to some other node,
% compute it there, and transmit the aswer back to host node.
%
% BEGIN MAIN LOOP
for i=1: size(t,2)

  % Store current queue lengths and percentage of problems calculated
```

```
  if size(tr_queue,2) > 0
    tr_queue_length(i) = size(tr_queue,1);
  else
    tr_queue_length(i) = 0;
  end

  if size(host_queue,2) > 0
    host_queue_length(i) = size(host_queue,1);
    host_queue_calc(i) = sum([host_queue.calc_left]);
  else
    host_queue_length(i) = 0;
    host_queue_calc(i) = 0;
  end

  for j=1: num_nodes
    if size(node_queue{j},2) > 0
      node_queue_length(j,i) = size(node_queue{j},1);
      node_queue_calc(j,i) = sum([node_queue{j}.calc_left]);
    else
      node_queue_length(j,i) = 0;
      node_queue_calc(j,i) = 0;
    end
  end
  subp_comp(i) = 100 * size(ans_list,1) / num_prob;


  % Incoming packet handling. When new problem packets have arrived,
  % compute_on_node -funtion is used to determine on what node the problem
  % will be calculated. The problem is then copied either to the host
  % computing queue or to the transfer queue to await transfer.
  if size(problems,2) > 0
    if problems(1).time_i <= t(i)
      problems(1).calc_node = compute_on_node(host_queue, tr_queue, node_queue,
problems(1));
      if problems(1).calc_node == 0
        problems(1).bytes_left = 0;
        problems(1).calc_node = 0;
        host_queue = [host_queue' problems(1)]';
        problems(1) = [];
      else
        problems(1).bytes_left = problems(1).bytes_desc;
        tr_queue = [tr_queue' problems(1)]';
        problems(1) = [];
      end
    end
  end


  % Transmit queue handling. If there are packages in the transmit queue,
  % trans * delta_t bytes of the first problem in the queue is transmitted
  % during the time step. If the transmission of the package is completed,
  % and the package is not an answer of the problem, problem is copied to
  % the computing queue of the node is has been transferred to. If the
  % completed package is an answer of the problem, the problem is copied
  % to the list of completed problems.
  if size(tr_queue,2) > 0
    tr_queue(1).bytes_left = tr_queue(1).bytes_left - trans * delta_t;
    if tr_queue(1).bytes_left <= 0
      tr_queue(1).bytes_left = 0;
      if tr_queue(1).calc_left ~= 0
        index = tr_queue(1).calc_node;
        node_queue{index} = [node_queue{index}' tr_queue(1)]';
        tr_queue(1) = [];
      else
        tr_queue(1).time_o = t(i);
        ans_list = [ans_list' tr_queue(1)]';
        tr_queue(1) = [];
      end

    end
```

```
  end


  % Host queue handling. If there are problems in the queue, comp_host *
  % delta_t operations of the first problem are calculated. If the first
  % problem is ready it is copied to the list of ready problems.
  if size(host_queue,2) > 0
    host_queue(1).calc_left = host_queue(1).calc_left - comp_host * delta_t;
    if host_queue(1).calc_left <= 0
      host_queue(1).calc_left = 0;
      host_queue(1).time_o = t(i);
      ans_list = [ans_list' host_queue(1)]';
      host_queue(1) = [];
    end
  end

  % Computation queue handling. If there are problems in the node
  % computation queue j, comp(j) * delta_t operations of the first problem
  % are calculated. If the first problem is ready, it is copied to the
  % trasfer queue to transmit the answer back to the host.
  for j=1: num_nodes
    if size(node_queue{j},2) > 0
      node_queue{j}(1).calc_left = node_queue{j}(1).calc_left - comp(j) *
delta_t;
      if node_queue{j}(1).calc_left <= 0
        node_queue{j}(1).calc_left = 0;
        node_queue{j}(1).bytes_left = node_queue{j}(1).bytes_ans;
        tr_queue = [tr_queue' node_queue{j}(1)]';
        node_queue{j}(1) = [];
      end
    end
  end

end
% END MAIN LOOP

% Drawing the results
subplot(2,2,1)
plot(t, subp_comp);
title('Percentage of subproblems completed');
xlabel('t [sec]');
ylabel('%');

subplot(2,2,3)
plot(t, tr_queue_length);
title('Length of transfer queue');
xlabel('t [sec]');
ylabel('subproblems');

subplot(2,2,2)
plot(t, [host_queue_length; node_queue_length]);
title('Lengths of computing queues');
xlabel('t [sec]');
ylabel('subproblems');
Names = [Name_host; Names];
legend(Names, 0);

subplot(2,2,4)
plot(t, [host_queue_calc; node_queue_calc]);
title('Lengths of computing queues');
xlabel('t [sec]');
ylabel('operations');

s1 = sprintf('\n%d out of %d subproblems processed', size(ans_list,1),
num_prob);
s2 = sprintf('Total time: %.1f\n', ans_list(size(ans_list,1)).time_o);
disp(s1);
disp(s2);
```

sim_constants.m

```
% ------------------------------------------------------------------------
% This .m -file is used to initialize parameters used in the simulation
% ------------------------------------------------------------------------


% Subproblems are structs with following fields:
%
% time_i        time when problem enters the system (sec)
% time_o        time when problem leaves the system (sec)
% bytes_desc    size of problem discription (bytes)
% bytes_ans     size of problem answer (bytes)
% bytes_left    size of problem not yet transferred (bytes)
% calc          operations required to solve problem (operations)
% calc_left     operations not completed (operations)
% calc_node     index of node calculating the problem

prob_type = struct('time_i',0, 'time_o',0, 'bytes_desc',0, 'bytes_ans',0,
'bytes_left',0, 'calc',0, 'calc_left',0, 'calc_node',-1);

t_start = 0;                        % start time of simulation (sec)
t_end = 1200;                       % end time of simulation (sec)
delta_t = 0.1;                      % length of one time step (sec)

num_prob = 500;                     % number of subproblems to be calculated

% Names and computing power of surrounding nodes (in operations / sec)
Names = {'Mobile 1'; 'Mobile 2'; 'Mobile 3'; 'Mobile 4'};
comp = [5.33e8 5.33e8 5.33e8 5.33e8]';
num_nodes = size(comp,1);           % number of surrounding nodes

% Name and computing power of host computer
Name_host = {'PC (host)'};
comp_host = 3200*1e6;

trans = 850 * 1024;                  % connection speed (bytes / sec)

k = 0.030031051*1e6;
```

compute_on_node.m

```
function fastest_index = compute_on_node(host_queue, tr_queue, node_queue, task)

% ------------------------------------------------------------------------
% This function determines on which node the problem 'task' takes shortest
% time to compute. The function does not take into account other packages
% arriving to the system at a later time.
%
% PARAMS:  host_queue  array for problems to be calculated on host
%          tr_queue    array for problems to be transmitted to other nodes
%          node_queue  arrays for problems to be calculated on nodes
%          task        new problem entering the system
%
% RETURNS: index of the node on which the computation takes shortest time
%          (0 if host is fastest)
% ------------------------------------------------------------------------

% Load constants
sim_constants;

% Calculate the time needed to compute the current problem on host computer
% if the problems currently in the host queue are calculated before this
% problem.
time_on_host = 0;
if size(host_queue,2) > 0
  time_on_host = (sum([host_queue.calc_left])) / comp_host;
end
time_on_host = time_on_host + task.calc / comp_host;


% Calculate the time needed to transmit the package to other nodes with the
% current transmit queue length.
time_to_transmit = task.bytes_desc / trans;
if size(tr_queue,2) > 0
  time_to_transmit = time_to_transmit + (sum([tr_queue.bytes_left])) / trans;
end

% Check if there are problems left to compute on different nodes when
% the current problem arrives there and calculate times to compute these
% problems. The current problems on the transfer queue are added to the
% corresponding computing queues.
time_on_node = zeros(num_nodes,1);
for i=1: num_nodes
  if size(tr_queue,2) > 0
    index = find([tr_queue.calc_node] == i);
    node_queue{i} = [node_queue{i}; tr_queue(index)];
  end
  if size(node_queue{i},2) == 0
    time_on_node(i) = 0;
  else
    node_comp_time = sum([node_queue{i}.calc_left]) / comp(i);
    time_on_node(i) = max([node_comp_time - time_to_transmit 0]);
  end
end

% Get the computation times of the task on different nodes and add these
% to node times
comp_times = task.calc ./ comp;
time_on_node = time_on_node + time_to_transmit + comp_times;

% Calculate the time needed to send the answer and answer of the other
% problems back to host computer
bytes_sent_back = 0;
for i=1: num_nodes
  if size(node_queue{i},2) > 0
    bytes_sent_back = bytes_sent_back + sum([node_queue{i}.bytes_ans]);
  end
end
return_times = zeros(num_nodes,1);
```

```
for i=1: num_nodes
  return_times(i) = max([(bytes_sent_back/trans)-comp_times(i) 0]);
end

% Add return times of other packages on the transfer queue before this
% package and time needed to transmit this package back to host
time_on_node = time_on_node + return_times + (task.bytes_ans / trans);

% Find the minimum of computing times and return it
if time_on_host <= min(time_on_node)
  fastest_index = 0;
else
  fastest_index = find(time_on_node == min(time_on_node));
  fastest_index = fastest_index(1);
end
```

generate_problems.m

```
function packages = generate_problems(p)

% ------------------------------------------------------------------------
% This function generates p problems used in simulation.m.
%
% PARAMS:  p    number of problems to be generated
%
% RETURNS: (p x 1) -array of structs of type 'prob_type' (defined in
%          sim_constants.m)
% ------------------------------------------------------------------------

% Load constants
sim_constants;

% Generate (p x 1) array of problems 'prob_type'
packages = repmat(prob_type,p,1);

% Generate sizes for the tasks. In the environment examined the size of
% problem description and problem answer are equal in bytes and all
% problems arrive to the system at time t = 0.
packages(1).time_i = 0;
packages(1).bytes_desc = (1024/5)*1024 + (1024/50)*1024*randn(1);
packages(1).bytes_ans = packages(1).bytes_desc;
packages(1).bytes_left = packages(1).bytes_desc;
packages(1).calc = k * packages(1).bytes_desc;
packages(1).calc_left = packages(1).calc;

for i=2: p
  packages(i).time_i = 0;
  packages(i).bytes_desc = (1024/5)*1024 + (1024/50)*1024*randn(1);
  packages(i).bytes_ans = packages(i).bytes_desc;
  packages(i).bytes_left = packages(i).bytes_desc;
  packages(i).calc = k * packages(i).bytes_desc;
  packages(i).calc_left = packages(i).calc;
end
```