Bluetooth Gaming with the Mobile Message Passing Interface (MMPI)

Daniel C. Doolan and Sabin Tabirca

University College Cork, Ireland

Abstract. The Mobile Message Passing Interface (MMPI) is a library implemented under J2ME to provide the fundamental functions that can be found in the standard MPI libraries used on Clusters and Parallel Machines. Nodes of a Cluster are usually connected to one another over a very high speed cabled interconnect. Within the mobile domain one does not have the luxury of connecting the devices with cabling, hence the MMPI library was built to take advantage of the Bluetooth capabilities that the majority of current mobile devices feature as standard. Mobile devices inherently have limited processing abilities. The MMPI library alleviates this problem by allowing the processing power of several devices to be used. Thus one can solve problems that a single device would be incapable of doing within a reasonable time frame. This paper discusses how the MMPI library can be applied to the application domain of Bluetooth enabled mobile gaming.

1 Introduction

The world of gaming is an ever expanding realm of the software sector. A myriad of console based game stations are available from the Playstation 2 and Playstation Portable to the X Box, and recently arrives Playstation 3. The mobile gaming market is also becoming ever more popular pass time for consumers. The market is driving phone users to constantly upgrade their phones and as such more and more people are purchasing high end phones capable of running games. The majority of phones support the execution of J2ME [1] applications. This however still leaves the developer with several problems such as varying screen sizes, and differences in how the virtual machine is implemented on across devices. Some of the API's may be device specific and the audio capabilities vary from model to model. The sales of Java based games are ever on the increase, this is driven by the fact that the process of installing a Java game is very straight forward. Mechanisms such as over the air provisioning (OTA) and WAP allow users to quickly and easily download the latest games from the content providers.

Games are generally classified into several genres that reflect the role of the gaming experience. These classifications may include: Strategy, Driving, Puzzle, Role Playing, Sports, Action and Adventure. Wang et. al [2] describe a new classification framework specifically for mobile peer to peer gaming. Within this classification they have broken down games into two distinct categories: Updating and User Interaction. The Updating dimension focuses on the updating of data between peers. This is divided into three

C. Doolan D. and Tabirca S. (2007).
Bluetooth Gaming with the Mobile Message Passing Interface (MMPI).
In *Proceedings of the 1st International Joint Workshop on Wireless Ubiquitous Computing*, pages 74-85
DOI: 10.5220/0002427400740085

subcategories: U1 Asynchronous, U2 Synchronous, U3 Real-time. The other dimension describes the type of user interaction, be it: I1 Controlled, I2 User Interaction, I3 Automatic Triggered or I4 Automatic. The main results of the work is that categories I1, I2 and U1, U2 are well supported under current Bluetooth enabled mobile devices.

1.1 Motivation

Originally the MMPI library was designed to enable parallel processing across several mobile devices connect together over a piconet network. All Bluetooth [3] [4] based applications have large sections of similar code (for example Device and Service Discovery). One must also manage a whole host of other details such as InputStreams and OutputStreams to achieve communication between devices. The MMPI library completely removes the need for writing Bluetooth specific code, hence one can focus on the topic at hand. As the library simplifies the inter-device communications process, it may be used for other purposes besides parallel computing applications. Thus it was decided to apply the library to the area of Bluetooth gaming [5].

Wang et. al [2] discuss that Real time updating of data over existing Bluetooth devices is not well supported. This paper describes how the MMPI library may be used to allow for the real time updating of multiplayer games as user interaction occurs across devices within the network.

The key contributions of this paper are the presentation of new data concerning the MMPI library. This relates to the metrics provided regarding the World creation times and the transmission times for communication. A template for the development of multiplayer games is presented, and a practical example of its usage discussed.

1.2 Key Terms and Background

The terms “World”, “Master” and “Slave” are used continually throughout this paper, therefore a clear understanding of these terms are critical to understand the context of this paper.

The “World” is used in standard MPI terminology to represent the communications world, this is the collection of nodes (processors) that make up the collective unit of the parallel machine. Every processor within the collective unit is capable of communicating with every node within the “World”.

Client / Server architectures use “Master” and “Slave” the differentiate between the controlling node and the worker nodes. Bluetooth technology is based on this architecture. The creation of a Bluetooth network of more than two devices will yield a Star network topology. In this form of network the “Master” is capable of communicating with any of the “Client” nodes. However, a Client device is unable to directly communicate with another Client device.

The MPI implementations found on Cluster, have an architecture where all nodes are equal. Every node is capable of communicating directly with every other node. Thereby the architecture is a fully interconnected mesh. In many MPI applications however a node called the “Root” node often controls the program execution. Parallel graphics is an excellent example of this where all of the actual drawing is carried out on the root node.

The MMPI system runs on top of Bluetooth, therefore the underlying technology is essentially Client / Server in nature. The MMPI system creates a fully interconnected mesh of all the devices in the "World", allowing each device to communicate directly with every other device. The only time that the Client / Server architecture is apparent is when an MMPI application is started, as it is necessary to define devices as Client and Servers to allow the Device Discovery process to execute. In MPI the manner in which the lines of communication are established is completely hidden from the developer.

1.3 Overview of Bluetooth Gaming

Bluetooth gaming is largely an under developed sector of the mobile gaming market. A chief reason for this is that only mid to high end mobiles come with Bluetooth as standard. Within this segment of the mobile market, many of these devices can have difficulty in communication because of differing implementations on the Bluetooth standard. Many devices do not fully support the standard, allowing for only a few devices to be connected rather than the eight defined as the maximum for a piconet network.

Mobile gaming is a very competitive market, therefore one must guarantee that any game developed is capable of running on as many devices as possible without problems to ensure maximum return on the capital invested in the project. Due to unreliability and the small market of Bluetooth devices, it is simply not technically of financially viable to develop multiplayer bluetooth games. However as with all technologies they are continually improving, and perhaps in the not too distant future multiplayer mobile games will rival the ferocious appetite for online multiplayer games that we see on the Internet today.

2 Mobile Parallel Computing

2.1 Message Passing

Since it's introduction in 1992 parallel programming has long benefitted from the Message Passing Interface (MPI). MPI is a library specification for message-passing, proposed as a standard by a broad group of vendors, implementers and users [6].

MPICH is one of the most well known freely available MPI implementations currently available [7]. MPI systems are usually based on the C or Fortran programming languages. Several Java based implementations are in existance (mpiJava) [8]. The mpiJava implementation is an object-orientated Java interface to the standard MPI libraries. It is implemented as a set of Java Native Interface wrappers to native MPI packages. An all Java example is Message Passing in Java (MPJ) [9] [10].

2.2 MMPI and Parallel Computation

The primary purpose of MMPI [11] was to produce an MPI like system for mobile devices. This abstracts the programmer from the Bluetooth communications libraries and allows for more productive development of the task at hand. The MMPI libraries speeds up computationally expensive tasks, by means of parallel processing. One example of

its usage in this regard is the computation Mandelbrot Set in parallel. This is a classical application of parallel computation [12], and is described as an "embarrassingly parallel computation" as each node can process a distinct section of the image without reliance on data from other nodes. An example of its usefulness may be seen in the fact that to generate a 200 pixel square Mandelbrot Set on a Nokia 6630 device requires almost one minute of processing. Generating the image with two nodes achieves a processing time of just over 30 seconds while four nodes allow for computation in approximately 22 seconds, using a Uniform Block approach.

Devices such as the Nokia 6630 and 6680 are powered by an ARM based processor running at 220Mhz [13]. This gives a computation capacity of approximately sixty million operations per second. With more powerful processors in constant development the mobile device will become the indispensable tool of the not too distant future, capable of more or less all the functions one would expect from a standard computer system. The announcement by ARM in October 2005 of their 1Ghz Cortex-A8 processor is a clear indication of the future of mobile computation [14] [15] [16]. Unlike previous ARM chips, the new Cortex-A8 has a superscalar architecture [17]

2.3 Initialising the World

Bluetooth itself is inherently a Client / Server based system. Therefore the underlying core of the MMPI library uses the same model. This is hidden from the developer who may be using the library. As with MPI, should it be necessary for one node to communicate with another, the Rank (identifier) of the node is the only variable that needs to be known.

The used of Bluetooth technology necessitates the need for Device Discovery, where by the Master (Controller) device discovers other devices within its catchment region. To create a World with four devices it is necessary to start three of them up in "client" mode before starting the "master" that will carry out the discovery process. In total their are three distinct phases in establishing a Bluetooth connection: Device Discovery, Service Discovery and the opening of Data streams.

The entire process of establishing a connection between two or more devices is a very slow process often taking twenty to twenty five seconds to complete. This is a detrimental factor to the uptake of multiplayer wireless gaming. People generally only play a mobile game for a few minutes, often while waiting for a bus, or while advertisements are on during a film on tv. Given that a game may last only two or three minutes having to wait for twenty five seconds or more just for the underlying communications system to be established is a huge percentage of the actual game play time. As an example a game may last for two minutes and require 25 seconds to initialse, therefore this accounts for almost 21% of the game playing time.

Of the three phases in establishing a connection Device Discovery takes the longest. According to the Bluetooth specification the inquiry phase must last for 10.24 seconds [18]. However in reality this figure is usually several seconds more. Scott et. al [19] discuss in their paper an alternative means of discovering devices by using Visual Tags. Under this scheme each device would have a small marker attached to the device. Using the built in camera one can retrieve an image of the tag which encodes a 48-bit Bluetooth Device Address (BD_ADDR) and 15 bits of application specific data. Thus one

need simply point a device to a tag to retrieve the device address. This greatly reduced the time it takes. Experiments showed the time to get the BD_ADDR with this new system was on average 0.56 seconds compared with 13.57 seconds for standard Device Discovery. The implementation was developed in C++. Another use of Visual Tags is from a company called OP3 [20] that offers a tool called ShotCodes that acts as an off line weblink. Once captured by the camera it will instantly take users to the desired location on the Internet.

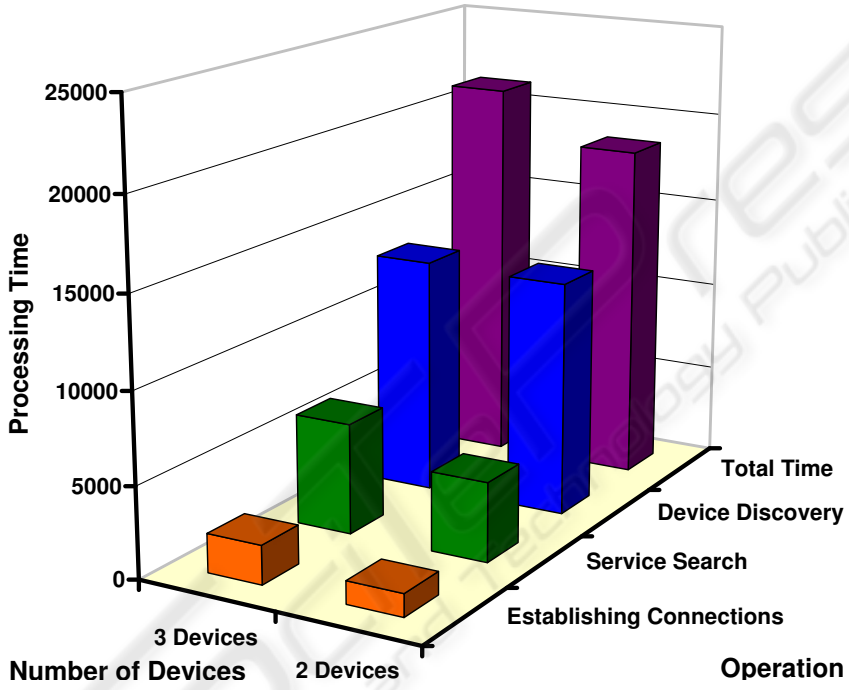

Fig. 1. World Creation Times for two and three Devices, processing time in milliseconds.

The Bluetooth standard specifies that a Bluetooth master can connect to seven Bluetooth slaves at the same time. This however seems to vary from device to device. Under tests carried out by Wang et. al [2] the Sony Ericsson P900 was capable of connecting to only one device the Siemens S65 could connect to three, while the Nokia 6600 was capable of connecting to seven devices as per the specification.

Figure 1 shows the times required to establish a World with both two and three devices. The tests were carried out with the devices in close proximity < 0.5 meters apart. A desktop machine was also within the Bluetooth range of the Master device. The discovery time in the case of the World with two devices required on average 12,019ms while the latter required 12,926ms. As expected the process of service discovery required some extra time for three devices over that of a World with two (6,109.5ms and

4,367.25ms respectively). The final stage of the process is the establishment of the data communication streams, the World with three devices required 2,136.5ms while the World of size two required 1,251ms. In the case of a World with two nodes the Master was a Nokia 6680 and the Slave a Nokia 6630. The world with three nodes again had the same device for the Master and both a 6680 and 6630 as the slave devices.

The creation of the data streams in the case of a World with three devices took significantly longer than that of the World with two, this is due to the establishment of additional Server and Client connections. This was necessary to create a fully connected mesh network as opposed to the standard Bluetooth style (star) network topology of a piconet. Hence the extra time represents the establishment of a Server connection on device 1 and the communication to device 0 that it has been established. As each Client can have many Server connections listening for Clients to connect, it is essential to synchronize the establishment of these connections. The first step is for device 1 to establish a Server connection. This device will then send a message back to the root node. Included in the message is the address of the newly created Server connection. The root will then forward the message to the required Client (Client 2). On receiving the message Client 2 can then open a connection to the Server of Client 1.

2.4 Communication

In MMPI Point to Point communication is carried out using the send(...) , recv(...) methods, in the same manner as MPI itself. The sending and receiving of data is carried out by specifying the device number to which data is to be sent, and the device number from where to data will be received from. In MPI programming a Send on one node must match up with a Receive on the destination node. Hence if you send some data from device 0 to device 1, device 1 must have a corresponding recv(...) method with device 0 specified as the device to receive from. Data is sent in the form of an Array of data which is in turn passed to the MMPI methods in the form of an standard java Object. The parameters required by these methods include: the input or output buffers, the starting position of the array, the number of elements to send or receive, the data type and the identifier of the node with which data will be sent to or received from.

One can achieve global communication with all nodes of the world by using a for loop and sending the data using the send(...) , recv(...) to each device. This is where collective communication methods are of use where by a node can for example send some data to all devices with just a single method call. This may be achieved by using methods such as Scatter, Gather and Broadcast.

A Logical Link Control and Adaptation Protocol (L2CAP) packet can have a maximum of $2^{16} - 1$ bytes of data payload [21]. Asynchronous Connection Less (ACL) links have a maximum payload size of 339 bytes [22]. The Radio Frequency Communication (RFCOMM) (used in the MMPI library) layer emulates RS-232 serial ports and serial data streams. RFCOMM relies on L2CAP for multiplexing multiple concurrent data streams and handling connections to multiple devices. The majority of Bluetooth profiles use this protocol due to it ease of use over direct interaction with the L2CAP layer.

The testing of the communication times between two devices was carried out in two differing manners. In the first example a flag is sent to the receiver device to indicate that it should get the start time and begin receiving the actual data. On receipt of all the

Table 1. Communication Times for send(...) , recv(...) , time is milliseconds, data size in Bytes, Tested using Nokia 6630's.

Data Size	Time
200	15
400	16
800	31
1,200	32
2,500	47
5,000	109
10,000	204

data the clock is again inspected and the communication time established. In this case the sending of data containing packet sizes of 4, 20 and 40 bytes all came back with a communication time of zero milliseconds.

Even for larger packet sizes the communication time is still very positive for example 200 bytes of data took 15ms. Even updating the graphical screen at a frame rate of 20fps, this still allows 50ms for the gathering of current data from other devices in the World. Table 1 shows the results for various packet sizes (up to 20000 bytes). The data being sent in these test cases were Integers hence 10 integers = 40 bytes. The Sender device was a Nokia 6680 and the receiver was a Nokia 6630.

Another set of tests that were carried out were of a more global nature where by data was sent to a receiver which in turn sent an acknowledgement back to the sender. This was tested using two (6680, 6630) and also three (6680 x2, 6630 x1) devices. Such a communication wouldn't typically be necessary for Bluetooth gaming, as generally one may just need to send the updated positions of a character for example. Table 2 shows the communications times.

In the case of two devices the 6680 was the Master node and the 6630 was a client node. For the World of three devices the 6680 was again the Master with the 6630 again being a client along with another 6680 device. Even for sending 40 bytes of data a time of 31ms still gives leeway for providing responsive graphics to all devices of the world. One may also note that the communication times for both two and three devices for data packets under 40 bytes are quite similar. These times are again the averages for several test runs.

Table 2. Communication Times for send(...) , recv(...) , time is milliseconds (Send + Ack), data size in Bytes.

Data Size	6630	6630 & 6680
4	31	32
20	32	31
40	32	32
200	47	63
400	47	109
800	47	141
1,200	52	187

2.5 MMPI for Bluetooth Gaming

The communications involved for the Bluetooth game (discussed in Section 3) requires the communication of the location of Keys, and the positions of Player Characters. In the case of a simple two player game one can use send(...) and matching recv(...) methods to achieve communication (Listings 1.2, 1.3). For true multiplayer gaming the use of global communication is necessary (broadcasting).

Listing 1.1 shows an of a typical communications Thread for the receiving of data. The first data to be received is simply a flag to indicate what type of data to receive next, be it character positions or key positions. For the recv(...) method to function they must be matched with send methods. Thus the sending of data happens in two distinct locations within the program.

```
for(;;){
  if(rank==0){
    mpiNode.recv(recvOperationType, 0, 1, MMPI.SHORT, 1);
    if(recvOperationType[0] == POSITIONMOVEMENT){
      mpiNode.recv(posBuffer, 0, 2, MMPI.SHORT, 1);
      coordX[1]=posBuffer[0];
      coordY[1]=posBuffer[1];
    }
  }else{
    mpiNode.recv(recvOperationType, 0, 1, MMPI.INT, 0);
    if(recvOperationType[0] == POSITIONMOVEMENT){
      mpiNode.recv(posBuffer, 0, 2, MMPI.SHORT, 0);
      coordX[1]=posBuffer[0];
      coordY[1]=posBuffer[1];
    }else{
      mpiNode.recv(recvKeyLocations, 0, 10, MMPI.SHORT, 0);
    }
  }

}
```

Listing 1.1. Example of recv(...) from within a Thread.

Sending of character positional information. When a user presses a key the keyPressed method is called. The position of the character on the users screen is updated, and consequently the other device needs to be informed of this position change. Therefore if the device is the Master then the positional information is transmitted to the Client, and vice versa.

```
protected void keyPressed(int keyCode) {
  if(rank==0){
    mpiNode.send(sendOperationType, 0, 1, MMPI.SHORT, 1);
    mpiNode.send(posCoords, 0, 2, MMPI.SHORT, 1);
  }else{
    mpiNode.send(sendOperationType, 0, 1, MMPI.SHORT, 0);
    mpiNode.send(posCoords, 0, 2, MMPI.SHORT, 0);
  }
}
```

Listing 1.2. Example of keyPressed().

The process of updating key positions is carried out by the controlling device (Master). Whenever the key positions are updated the new positions must be transmitted to the Client device so both game screens remain synchronised.

By combining the sending operations of both the updateKeys() and keyPressed() methods the data transmission operations are perfectly matched with the corresponding

receive operations carried out within the receive Thread. The example listings above demonstrate that only a small amount of data is transmitted at any instant. This allows for a very short latency between transmission and receipt of the data. In the case of positional information an array of size two is sent. This occurs at every keyPressed event. The sending of an array of size ten is carried out every second for the updating of key positions.

```
updateKeys(){
  setNewKeyPositions();
  mpiNode.send(sendOperationType, 0, 1, MMPI.SHORT, 1);
  mpiNode.send(sendKeyLocations, 0, 10, MMPI.SHORT, 1);
}
```

Listing 1.3. Example of updateKeys().

2.6 MMPI Gaming Template

Many games have a similar structure. They all need screens / canvas's for the Game Setup and the Game Conclusion. A GameThread class may be provided to run the game animation. Thus the only classes that need significant modified are the class directly dealing with the display of the graphics. These may be summarised as GameCanvas and GameLayerManager classes. By combining the MMPI system with this structure one can achieve a template that allows for a wide selection of games to be rapidly developed (Figure 2).

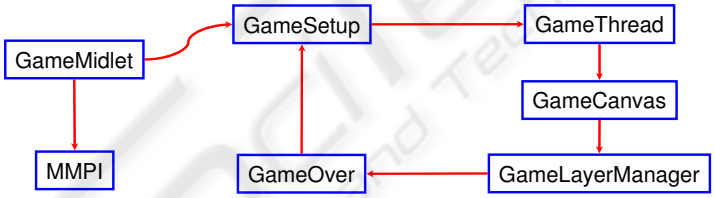

Fig. 2. Template for MMPI Gaming.

3 The Bluetooth Game

The game that was implemented with the library was that of a maze game. To play the game each player must first select the appropriate mode to play the game in be it Master or Client. The next few screens allow the user to choose a character and and give the character some attributes (Figure 3). Once the character has been fully created the game itself may be played. The objective of the game is for the user to guide their character through the maze and get to the end point (the room with the treasure chest). The player who reaches the treasure chest first is declared the winner. To complete a level users must collect keys so they can pass through the doors and into the adjoining room. To

win a level it is necessary to have accumulated five keys, with which the treasure chest may be unlocked and the level concluded.

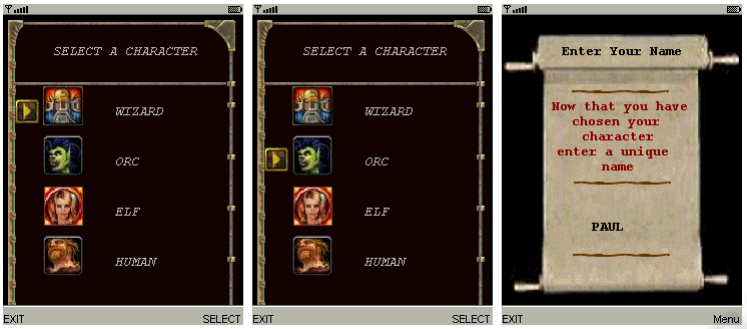

Fig. 3. Initial Character Configuration Screens.

Two distinct properties must be communicated between the devices. These comprise the positions of the characters, and the positions of the keys (Figure 4). The Master device controls the position of the keys which are generated randomly. Each key staying in the same position for 5 seconds before moving.

Fig. 4. The Maze Displayed on two Devices.

The key positions are changed every second hence the new positions must be communicated to the Client devices. The most time critical values are that of the positions of the Characters, as it is necessary that the characters appear at the same location on all the devices the moment a player moves. Therefore the moment a player moves a character, the positional information must be transmitted to all the other devices within the World.

In the initialisation phase of the game it is necessary for all devices to tell every other device what character has been chosen so the correct characters can be displayed on screen. At the conclusion of a level a message must be sent to all the devices to indicate that somebody has won the level, hence prepare to receive new random positions from the master device.

4 Conclusion

This work has shown that the MMPI library is suitable not only for Parallel programming over Bluetooth networks, but that it can also be successfully applied to the domain of wireless gaming on mobile devices. It has been shown that one can use the MMPI library to produce a game that provides up to date graphics to each of the users playing the game. To ensure all user devices display the exact same screens it is essential that the amount of data transmitted is kept to a minimum so that latency issues in the updating of the graphical displays are kept to a minimum. Data has been presented to show the creation time of an MMPI World, and the communication times for a varying selection of data transmission sizes. In summary the MMPI library provides the developer with a very simple to use interface for the development of Bluetooth enabled games, without the need to develop any Bluetooth specific code themselves.

Acknowledgements

Research funding source: ”Irish Research Council for Science, Engineering and Technology”, funded by the ”National Development Plan”. We would also like to thank Tracey J. Mehigan for her work on the Bluetooth game.

References

1. Microsystems, S.: Java 2 platform, micro edition (j2me). http://java.sun.com/j2me/index.jsp (2007)
2. Wang: Issues related to development of wireless peer-to-peer games in j2me. In: International Conference on Internet and Web Applications and Services (ICIW 20006), IEEE Computer Society Press (2006)
3. Bluetooth.com: The official bluetooth website. http://www.bluetooth.com/ (2007)
4. Bluetooth.org: The official bluetooth membership site. http://www.bluetooth.org/ (2007)
5. Long, B.: A study of java games in bluetooth wireless networks. Master’s thesis, Department of Computer Science, University College Cork (2004)
6. MPI: The message passing interface (mpi) standard. http://www-unix.mcs.anl.gov/mpi/ (2007)
7. MPICH: Mpich - free implementation of mpi. http://www-unix.mcs.anl.gov/mpi/mpich/ (2007)
8. HPJava: mpijava. http://www.hpjava.org/mpiJava.html (2007)
9. DSG: Message passing in java (mpj) project. http://dsg.port.ac.uk/projects/mpj/ (2007)

10. M. Baker, D.C.: Mpj: A proposed java message-passing api and environment for high performance computing. In: In Proceedings of IEEE International Parallel & Distributed Processing Symposium. (2000)
11. Doolan, D.C., Tabirca, S.: Mobile parallel computing. In: Fifth International Symposium on Parallel and Distributed Systems. (2006) 161–167
12. Book, M.: Parallel fractal image generation. http://www.mattiasbook.de/papers/parallelfractals/ (2007)
13. Freak, S.: Nokia 6680 is loosing the battle to 6630. http://www.symbian-freak.com/news/0305/6680.htm (2006)
14. ARM: Arm cortex-a8. http://www.arm.com/products/CPUs/ARM_Cortex-A8.html (2005)
15. ARM: Arm introduces industry's fastest processor for low-power mobile and consumer applications. http://www.arm.com/news/10548.html (2005)
16. Pilato, F.: Arm reveals 1ghz mobile phones processors. http://www.mobilemag.com/content/100/102/C4788/ (2005)
17. Robinson, D.: Arm chips to power 1ghz mobiles. http://www.vnunet.com/itweek/news/2143741/arm-chips-power-1ghz-mobiles (2005)
18. Bluetooth-SIG: Annex a (normative): Timers and constants. In: Bluetooth Specification Version 1.1. (2001)
19. Scott, D.: Using visual tags to bypass bluetooth discovery. In: ACM Mobile Computing and Communications Review (MC2R). (2005)
20. OP3: Shotcodes. http://www.op3.com/en/technology/shotcodes (2007)
21. Klingsheim, A.: J2me bluetooth programming. Master's thesis, Department of Informatics, University of Bergen (2004)
22. Nokia: Games over Bluetooth: Recommendations to Game Developers. Nokia (2003)