Advances in Engineering Science
Sect. C (2), April-June 2008, PP 37-46

OPTIMIZED FTP

Mr. Mahajan S. A.[1]*, Mr. Patil S. H.[1], Mr. Khadtare M.[2]

1- B.V.D.U.College Of Engg. Computer Dept.Pune, India
2 - IIT, Guwahati, India

ABSTRACT

FTP is file transfer protocol is basically to transfer for large volumes of data. Implementations of it can be widely deployed and can be used on wellconnected network because of its ability to scale to network speeds. We propose the optimization technique to improve the performance of FTP[1], measured performance using the various profilers. This Profile data is valuable for identifying performance bottlenecks and guiding optimizations. The FTP ported on various hardware platforms such as P-III, P-IV with MMX, SIMD architecture based. We have use deoxygen/ system clock (gettime()) tool techniques for gathering and manipulating profile information at varying degrees of precision, particularly in the presence of various optimizations techniques such as inlining, c level optimization, loop unrolling, intrinsic, utilization of pipeline stage for processors with compiler level coding. We found that with various levels of optimization stages we achieve that data memory and program memory saving with effect of 60% of actual size not affect the actual performance. As FTP contains the compute intensive modules such as communicating protocol as ISO-OSI[2] layer to transfer data it affects the bandwidth and processing speed of CPU core. This method gives us performance nearer to GridFTP high performance computing model (Note: GridFTP used mostly HPC processor to measure its work[3]).The aim is to provide the benchmark for the furher research in this diection, so that it could be applicable to the mobile decives for efficient handling of data memory. The benchm could be useful to acheive the space and speed optimization.

Index Terms : *Data transfer, small files, FTP, profilers, Secure data transfer, Parallel streams.*

1. INTRODUCTION

Sharing of information is essential for organizations today. Internet serves this purpose because it has the ability to move files. FTP is a well known protocol used for uploading and downloading files from Internet. It is ddefined in RFC-959 [1][4]. The aim is to show results which can make the FTP fast and efficient through optimization. The protocol is optimized to transfer large volumes of data,also to utilize minimum space and achieve faster data transfer considering ideal network connection. Given the high-speed networks commonly found in modern grid environments, datasets less than 100 MB are too small for the underlying protocols like TCP to utilize the maximum capacity of the network. Therefore, FTP[5] and most bulk data transfer protocols experiences the highest levels of throughput when transferring large volumes of data.

The typical quality of service requirements i.e. transfer delay, throughput rates for high speed protocols impose strong performance requirements on high speed protocol implementations. As the throughput of the networks has increased much faster than the processing power of processors these requirements can only be satisfied by efficient processing of protocol data by the involved protocol machines. Different approaches to improve the performance of communication protocols have been proposed .It could be done by the improvements by changes to the protocol mechanisms and hardware implementation of protocol functions and by parallelizing the implementation of communication protocols. This paper suggests to optimize the FTP protocol functions over multiple processors like P-III and P-IV with either dedicated or general purpose

*sa_mahajan@yahoo.com

functionality, thus an SIMD parallelization. We will focus on this parallelization approach.

Periodic sampling of a processor's performance monitoring hardware is an effective, unobtrusive way to obtain detailed profiles. Unfortunately, existing hardware simply counts events, such as cache misses and branch mispredictions, and cannot accurately attribute these events to instructions, especially on out-of-order machines. We propose an alternative approach, with deoxygen tool, that samples instructions. As a sampled instruction moves through the processor pipeline, a detailed record of all interesting events and pipeline stage latencies is collected.

Our optimization will support paired sampling, which captures information about the interactions between concurrent instructions, revealing information about useful concurrency and the utilization of various pipeline stages while an instruction is in flight. We describe an inexpensive software implementation of our optimization technique, outline a variety of software optimization techniques to extract useful profile information from the hardware. This information can provide valuable feedback for optimization of FTP. We had carried out work on P-IV with 3.00Hz @2.99GHz,248MB RAM, P-IV 2.66 GHz @2.67GHz and P-III x-86 family, 533 MHz,127 MB RAM.

2. Related Work

Users often intend to put all the data into a single file and then they plan to send or transfer that single file. This process requires additional CPU time and disk space. The Grid FTP Pipelining taken by[3] is for the Grid environments. We had worked on similar lines but by using various profiling techniques and the machines of Pentium level which are used in large scale. With this we had optimize the performance of our FTP.For this purpose we had refeered FTPWanderer1.2.0 [15] version of the FTP program. This allows us for many file transfers to occur at the same time. The aim here is to reduce the size of executable file in order to achieve the space optimization which is very important aspect in the mobile devices.This is because now we use these mobile devices widely for the transfer of data..For this devices the speed and space matters. So we are aiming to minimize the size of application program which is going to recide on these mobile devices also on the lower level Pentium machines,which are still are in use widely. The approach we present here has the significant potential.

3. Network Protocol Issues

Transfer of data can be done with either TCP or UDP depending on the need and application. We choose TCP over UDP due to various reasons. The reasons are TCP provides full featured protocol that allows applications to send data reliably without worrying about network layer issues, is a connection oriented protocol, reliable delivery, more scalable and adapts to growing as well as congested networks, it can send about 8-12 segments at a time before waiting for an acknowledgement instead of UDP sending 1 segment then waiting for an ACK has an effect on the performance. So the standard protocol for network data transfer remains TCP. FTP is a file transfer system, is part of TCP/IP suite[6][7][8]. However, TCP's congestion avoidance algorithm can lead to poor performance, particularly in default configurations and on paths with high round trip times. Solutions to this problem include careful tuning of TCP parameters, TCP protocol improvements, multiple "parallel" TCP connections, and the substitution of alternative protocols

FTP is a widely implemented and well understood standard protocol with a large base of code and expertise from which to build. Secondly, FTP provides a well-defined architecture for protocol extensions and supports dynamic discovery of the extensions supported by a particular implementation. Apart from the common design objectives of the FTP importantly, RFC959 also notes that FTP, whilst being directly usable by the user, is designed mainly for use within programs, i.e. a program provides an easy interface through which the FTP protocol may be used.

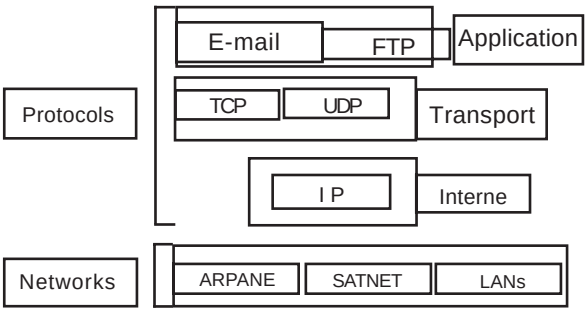

Figure 1

FTP[9] maintains the same command/response semantics introduced by RFC959. It also maintains the two-channel protocol semantics. One channel is for control messaging (the control channel) such as requesting what files to transfer, and the other is for streaming the data

pay load (the data channel). These protocol details have interesting effects on the optimization problem. In FTP Access control is normally accomplished by associating a number of access flags with each file and directory(e.g. a read-only flag).How this is done is OS specific.

Three groups of access flags are provided: for user, their workgroup and then general access. Three flags are included within each group: one for read access, another for write access, and a third for execute privileges.

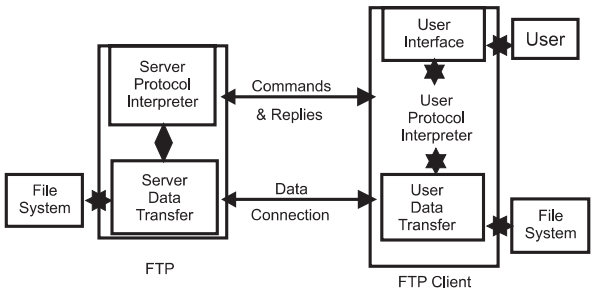

Figure 1

The FTP comprises three logically distinct components: client and server protocol interpreters (PIs), which handle the control channel protocol (these two functions are distinct because the protocol exchange is asymmetric), and the data transfer process (DTP), which handles the accessing of the actual data and its movement via the data channel protocol. These components can be combined in various ways to create servers with different capabilities. For example, combining the server PI and DTP components in one process creates a conventional FTP server, while a striped server might use one server PI on the head node of a cluster and a DTP on all other nodes.

4. Channel Establishment

4.1File Transfers

FTP servers listen on a well-known and published port for client control channel connections. Once a client successfully forms a control channel with a server (this often involves authentication and authorization), it can begin sending commands to the server. In order to transfer a file, the client must first establish a data channel. This involves sending the server a series of commands on the control channel describing attributes of the desired data channel such as: what protocol to use, binary or ASCII data, passive or active connection, and various protocol specific attributes. Once these commands are successfully sent, a client can request a file transfer.

At this point a separate data channel connection is formed using all of the agreed upon attributes and the requested file is sent across it. In standard FTP the data channel can be used only to transfer one file. Future transfers must again go through the process of setting up a new data channel. Through our program FTP modified this part of the protocol to allow many files to be transferred across a single data channel.

With FTP we had tried for all of the messaging to establish a data channel is done once; the data channel connection is formed just once, and the client can request several file transfers using that same data channel. This enhancement is called as data channel caching.

File transfer requests are done with the RETR (send) or STOR (receive) command. A client sends one of these commands to the server across the

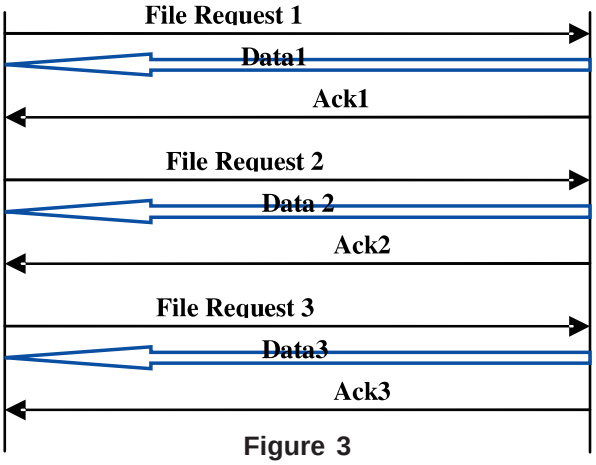

Figure 3

control channel. Data then begins to flow between the client and server over the data channel. Once all of the data has been transferred, a "Transfer Complete" acknowledgment message is sent from the server to the client on the control channel. Only when this acknowledgment is received can the client request another transfer. This interaction is shown in Figure 4. As the figure shows, there is an entire round-trip time on the control channel between transfers where the data channel must be idle. Before issuing the next transfer command the client must first receive the transfer completion acknowledgment, which is one trip across the network. After receiving the acknowledgment, the client sends the transfer command immediately. However, the server does not immediately receive it. The message must cross the network before the server will begin sending data. This process involves

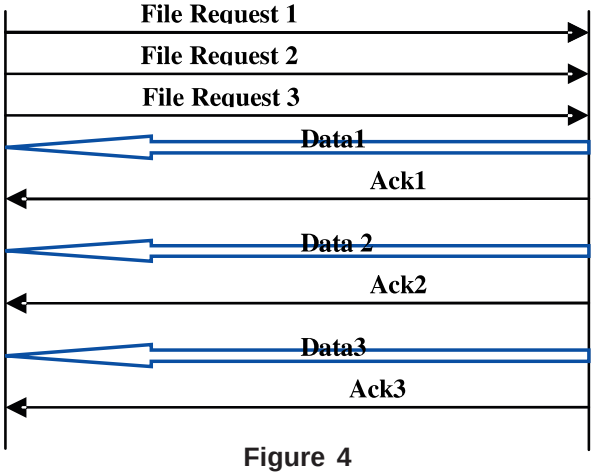

Figure 4

another trip across the network. Assuming we have the FTP data channel caching enabled, we do not have to worry about the latencies involved with establishing the data channel. If we do not have it enabled, the delay is significantly longer.

During this time the data channel is idle. The latency between transfers adds to the overall transfer time and thus detracts from the overall throughput. The problem can have high implications when communicating over high latency networks where the RTT is very high. While the idle data channel time is a problem, there is a far greater problem that it causes.

TCP is a window-based protocol. For it to achieve maximum efficiency, the window size of allowed unacknowledged bytes must grow to the bandwidth delay product. Various algorithms in the TCP protocol decide to increase or decrease the window size based on observed events . If a connection is idle for longer than one RTT, the window size gets reduced to zero; and once it is used again, it must go through TCP slow start.

When transferring a series of files, the data channel is idle for a control channel RTT in between transfers. If the control channel RTT and the data channel RTT are similar, it is likely that data channel TCP connections will have entire closed windows by the time the next transfer begins. When the amount of data sent in each file is small, the ratio of idle data channel time to transfer time becomes higher and affects the throughput. Additionally, small files may not be transferred long enough to traverse the slow-start algorithm and bring TCP to full throttle. Thus, even when data is being transferred, it is not moving at full speed.

5. PROFILING

Given the relatively large number of high-performance transfer tools, the question about the effectiveness of each of them arises naturally. However, although prototypes of many of the systems have been around for a while, an experimental comparison is still lacking in the literature. This paper aims at filling this gap by presenting the results we collected by performing data transfer experiments, among machines which

are pentium compatible, using some of the tools mentioned here. We tried to answer the following questions,

1. Need of profiling?

The first stage of any optimization process is to identify the critical routines and measure their current performance. A profiler is a tool that measures the proportion of time or processing cycles spent in each subroutine. We use a profiler to identify the most critical routines. A cycle counter measures the number of cycles taken by a specific routine. We had measure our success by using a cycle counter to benchmark a given subroutine before and after an optimization. Profilers [10]use a wide variety of techniques to collect data, including hardware interrupts, code instrumentation, operating system hooks, and performance counters. The usage of profilers is called out in the performance engineering process.

2. Improvement in the optimization [10]

1) Space optimizations -Reduces the size of the executable/object.
   1) Constant pooling
   2) Dead-code elimination.
   2) Speed optimizations. Most optimizations belong to this category. There are important optimizations not covered above, e.g. the various loop transformations:

1) Loop unrolling -Full or partial transformation of a loop into straight code. Eliminating the loop and writing code separately for each loop index Significantly increases speed.

```
for (int x=0; x < 100; x ++)
{
delete(x);
}
```

If this part of the program is to be optimized, and the overhead of the loop requires significant resources, loop unwinding can be used to speed it up. This will result in an optimized code fragment like:

```
for(int x=0; x < 100; x += 5)
{
delete(x);
delete(x+1);
delete(x+2);
delete(x+3);
delete(x+4);
}
```

2) Loop blocking (tiling) -Minimizes cache misses by replacing each array processing loop into two loops, dividing the "iteration space" into smaller "blocks".

3) Loop interchange -Change the nesting order of loops, may make it possible to perform other transformations.

4) Loop distribution -Replace a loop by two (or more)equivalent loops.

5) Loop fusion -Make one loop out of two (or more) equivalent loops.

6) inlining-This is an efficient language-independent optimization technique. We have used the inline function in combination with the options like maximum speed,minimum size,global optimization and full optimization. Function calls involve changing the stack pointer and program counter registers, passing arguments, and allocating space for results. This means almost the entire state of the CPU itself is saved then restored at each function call. It is effective on highly pipelined CPUs.

3. Optimizing the code and program memory with c level techniques.

In general design changes tend to affect tperformance more than "code tweaking".

Here we had attempted simple mathematical analysis. We calculate the approximate running time of our algorithm (i.e., calculate its "**O**") [12]taking all bottlenecks into account like is it optimal? can we prove it? can we justify up our algorithmic design with theoretically known results? For our purpose we have used ‘O’ level ,Loop unrolling,Inline function and intrinsic level optimizations.The ‘O level optimization controls the overall level of optimization. This makes the code compilation take somewhat more time, and can take up much more memory, especially as we increase the level of optimization.There are five -O settings: -O0, -O1,-O2,-O3,-O4. With Inline function it Reduces the call overhead and identify the subprograms that are called the most and inline only

those subprograms. An example of inlining via macros: Old code:

```
int foo(a, b)
){
{
a = a -b;
;b++;
;a = a * b;
;return a;
;}
}
```

New code:

```
#define foo(a, b) (((a)-(b)) *
*((b)+1)).
```

.The for loop unrolling or loop punwinding is already explained d

here.Use intrinsic functions. Intrinsic functions are C like functions that directly map to the low level instructions of the CPU. Often the use of these functions allow specific or more efficient variations of standard mathematical operations (E.G. +-/x).

This will significantly improves the CPU performance. We carried out the results on Pentium platform.Now the question is why the Pentium platform?The reason is these are in use widely.Although we have the Pentium machines available which are much faster and efficient and propably don't require the optimization.But in order to achieve the higher throughput on the P-III machine we need to optimize the application program like FTP to achieve fast data transfer rate,to have space utilization,to achieve memory utilization. The Pentium architecture[10] has, 1)Instruction Prefetcher: There are now a total of four prefetch buffers, each of them 32 bytes wide (it's going to turn out later, when we talk about caches, that this is the Right Size object to fetch from cache on each transfer); they work in conjunction with the Branch Target Buffer.

Normally, one pair of prefetch buffers is actively grabbing instructions; if the BTB predicts a branch as taken, the other buffer will start fetching instructions from the branch target. Later on, if the prediction is wrong, the pipelines will have to be flushed. 2)Decode 1: This is where the pipelines split. Two parallel decoders attempt to decode and issue the next two sequential instructions. The decoders determine whether one or two instructions can be issued, depending on the pairing rules. Registers are also read in this stage. 3)Decode 2: Memory operand addresses are generated here. We can infer that there are either two adders for address generation, or a single three-operand adder (such a beast can actually be built).

4)Execute: Just like 486, except that all instructions except conditional branches are verified for correct branch prediction (so we don't write a result back to registers on a mispredict). 5)WB: Just like 486, except that conditional branches are verified for correct branch conditions (again, we don't want to write results back to memory on a mispredict).

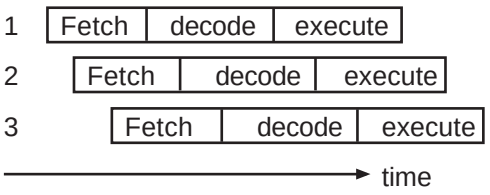

Figure 5.

As we are porting our FTP on P-III and P-IV machines ,we had used this pipeline info to achieve desired performance. We also used the info of SIMD implementation to reduce the iterations through loop unrolling and deoxygen tool[11].The SIMD concept is a method of improving performance in applications where highly repetitive operations need to be performed.

Simply put, SIMD[11] is a technique of performing the same operation, be it arithmetic or otherwise, on multiple pieces of data simultaneously.

Ideally, to increase performance, the number of iterations of a loop needs to be reduced. Once method of reducing iterations is known as loop unrolling. This takes the single operation that was being performed in the loop, and carries it out multiple times in each iteration. For example, if a loop was previously performing a single operation

and taking 10,000 iterations, its efficiency could be improved by performing this operation 4 times in each loop and only having 2500 iterations.

The SIMD concept takes loop unrolling one step further by incorporating the multiple actions in each loop iteration, and performing them simultaneously. With SIMD, not only can the number of loop iterations be reduced, but also the multiple operations that are required can be reduced to a single, optimized action.

6. IMPLEMENTATION

Two important questions to ask when tuning software are: (1) how to identify what code to focus on, and (2) how to estimate the benefit of recoding, and/or re-compiling with an optimized compiler? A beneficial approach for getting answers to these questions is to sort the execution times of a given workload into sections according to the amount of time spent in each section of the executed code. By focusing on small sections of code that consume greater proportion of execution time and using an accurate tool for measuring performance improvement, the challenge of estimating the reward of optimizing an application becomes easier. Combined with an accurate tool for estimating likely application performance gain for each coding situations, this can ensure software tuning effort is focused on the primary coding issues. In first run we have note down the .exe size i.e. FTP Wanderer.exe. In second run we applied the 'O' level optimization.At third run we applied inbuilt Inline function.At fourth run we applied for loop unrolling technique.All the results were tabulated.For the desired result we applied the complier or intrinsic level optimization.

7. RESULTS

Actual performance results on target applications will be influenced by many factors, ranging from the workload characteristics of the application, implementation details of the re-coding effort, hardware and software configurations, etc. The approximate ranges of likely performance gains are based on a comparison of performance results between a typical Pentium 4 processor platform relative to a typical Pentium III processor platform, with similar hardware configurations and with the frequency of the Pentium 4 processor running at approximately 1.5X higher than that of the Pentium III processor.

The profiler tool[13][14] used here is useful for identifying critical code paths and performance bottlenecks. For example, it can be used to sample and compare performance data when the application to be optimized is run on two different target processors; for example, a Pentium 4 processor running at 1.5 GHz and a Pentium III processor running at 1 GHz. This performance data from the two targets can be sorted and displayed at different scopes ranging from modules to functions, to assembly code. This capability allows us to identify individual modules, and individual functions as "hot spots". We had carried out work on two P-IV machine and one P-III machine. The following table shows the results with general category and warning levels rangining from O1 to O4 optimization. The complier speed varies from default to the maximum speed. The original exe file size and the final size after optimizationis shoen in tabular format here.The results may vary at each subsequent run. The results shows the variations when we change the optimization levels as given in the table,

Table 1-is with the'O' lecel optimization i.e. from 'O1' to 'O4'.Table 2 –is woth inline function. with various options. Table 3 – is with loop unrolling and Table 4 – is with intrinsic level optimization.

Optimization Level	None	1	2	3	4
Optimization Speed	default	max. speed	max. speed	max. speed	max. speed
Results in kb with inline function					
P-IV, 3.00GHz, 2.99GHz 248MB RAM	392 Kb	384 kb	372 kb	372 kb	348 kb
P-IV, 2.66GHz, 2.67GHz 448MB RAM	392 Kb	372 kb	362 kb	350 kb	340 kb
P-III,x -86 family,533 MHz,127 MB RAM	393 Kb	382 kb	373 kb	373 kb	352 kb

Table 1.'O' level

Category	Optim -ize	Optim -ize	Optim -ize	Optim -ize	Optim -ize
Optimization Speed	default	max. speed	min size	global	full
Results in kb with inline function					
P-IV, 3.00GHz, 2.99GHz 248MB RAM	420 kb	392 kb	390 kb	384 kb	384 kb
PIV, 2.66GHz, 2.67GHz 448MB RAM	420 kb	390 kb	375 kb	362 kb	362 kb
P-III, x-86 family,533 MHz,127 MB RAM	501 kb	465 kb	457 kb	393 kb	369 kb

Table 2.a)With Inline expansion - Disable

Category	Optim -ize	Optim -ize	Optim -ize	Optim -ize	Optim -ize
Optimizatio n Speed	default	max. speed	min size	global	full
Results in kb with inline function					
P-IV, 3.00GHz, 2.99GHz 248MB RAM	420 kb	395 kb	390 kb	384 kb	384 kb
PIV, 2.66GHz, 2.67GHz 448MB RAM	420 kb	390 kb	370 kb	362 kb	360 kb
P-III, x-86 family, 533 MHz,127 MB RAM	497 kb	493 kb	473 kb	465 kb	465 kb

Table2. b) With Inline expansion - Inline

Category	Optim -ize	Optim -ize	Optim -ize	Optim -ize	Optim -ize
Optimizatio n Speed	default	max. speed	min size	global	full
Results in kb with inline function					
P-IV, 3.00GHz, 2.99GHz 248MB RAM	420 kb	396 kb	392 kb	390 kb	384 kb
PIV,2.66GHz, 2.67GHz 448MB RAM	420 kb	390 kb	375 kb	364 kb	362 kb
P-III,x-86 family,533 MHz,127 MB RAM	497 kb	493 kb	477 kb	473 kb	473 kb

Table2.c) With Inline expansion - Any suitable

Optimizati on Level	None	1	2	3	4
Optimizati on Speed	default	max. speed	max. speed	max. speed	max. speed
Results in kb					
Dual Core,Intel-Core(Pm)2 BUoCPU, D4500@2. 20GHz,2.4 8MB RAM,Phys ical address extension	4162 Kb	388 kb	384 kb	384 kb	364 kb
P-IV, 2.66GHz, 2.67GHz 448MB RAM	408 Kb	400 kb	390 kb	384 kb	384 kb
P-III,x-86 family,533 MHz,127 MB RAM	393 Kb	382 kb	373 kb	373 kb	352 kb

Table 3 For Loops Unrolling

Optimization Level	None	1	2	3	4
Optimization Speed	default	max. speed	max. speed	max. speed	max. speed
Results in kb					
P-IV, 3.00GHz, 2.99GHz 248MB RAM	392 Kb	372 kb	353 kb	322 kb	322 kb
P-IV, 2.66GHz, 2.67GHz 448MB RAM	384 Kb	362 kb	340 kb	329 kb	328 kb
P-III, x-86 family, 533 MHz,127 MB RAM	393 Kb	384 kb	363 kb	352 kb	340 kb

Table 4 : For Intrinsic level

To determine whether a prominent hot-spot module or function is a cause of poor performance, the sampled data from profiler can be further processed for comparison based on a relative performance scaling factor between the two target processors. Typically, those modules (or functions) that represent performance bottlenecks are identified by a relative scaling factor, that falls significantly below 1.0 or another known characteristic of the workload. The table shows that we got better results with the optimization levels ranging from 01 to 04 comparing to inline function.

8. CONCLUSION AND FUTURE WORK

This experiment is to study the optimization and to optimize the performance of a program specifically the network program which incorporates the network bandwidth, processors speed, protocols used etc. Through these experiments we had presented a solution to optimize the performance of FTP system using Profiling concept. Our results show that the profiling and optimization approach is effective for slower as well as faster processors at P-III and P-IV level. The results may vary a bit depending on the processors speed. The results has the revalence to those working on Pentium level machines. These results certainly will provide the benchmark for the further research in the direction of optimization of application program for the mobile devices. Although we have achieved good results with this kind of processor speed, we plan to carryout the further research work in direction of optimization of program for mobile devices and for High performance computing (HPC) network solutions.

9. REFERENCES

1. J. Postel, J. Reynolds, File Transfer Protocol (FTP). RFC 959, Internet Engineering Task Force, October 1985.
2. J. Postel, Transmission Control Protocol. RFC 793, Internet Engineering Task Force, September 81.
3. John Bresnahan,Michael Link, Rajkumar Kettimuthu,Dan Fraser, Ian Foster. Grid FTP Pipling, Teragrid Conference, Madison, W, 2007
4. M. Allman, V. Paxson, W. Stevens, TCP Congestion Control. RFC 2581, Internet Engineering Task Force, April, 99.
5. The Case for Secure File Transfer: Overview of GlobalSCAPE's Enhanced File Transfer (EFT) Solution.Published: July 2005 .
6. Mastering IIS FTP - Part 2 - Virtual Directories-Physical Directories - Scott Forsyth's WebLog.
7. www.cute.FTP.com
8. www.gnu.org/software/tar
9. The bbFTP-Large Files Transfer Protocols Websitewww.doc.in2p3.fr/bbftp
10. Desktop Performance and Optimization for Pentium 4 Processor, www.intel.com/procs/perf/pentium4 .

11. Jeffrey Dean,Jmes E. Hicks,Carl A.Waldspurger, William E. Weihl, George Clurysos, Hardware support for instruction level profiling on out-of-order processors. Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, 1997, PP 292-302

12. David Grove, Jeffrey Dean, Charles Garrett, Craig Chambers. Profile-guided receiver class prediction, ACM SIGPLAN Notices, Volume 30, 1995, PP 108-123.

13. Karl Pettis, Robert C. Hansen, Profile guided code positioning, ACM SIGPLAN Notices, Volume 25, 1990, PP 16-27.

14. Merten, M. C. Trick, A. R. George, C. N. Gyllenhaal, J. C. Hwu, W.W., A Hardware-driven profiling scheme for identifying program hotspots to support runtime optimizations, Computer architecture, Proceedings of 26th International Symposium, 1999, PP 136-148.

15. http://www.pablovandermeer.nl

■