Bahria University
Discovering Knowledge

P&DC

ASSIGNMENT 1

Submitted by:
Syed Asmar Hasan - 006

Department Of Computer Science
Bahria University, Karachi Campus

Q1. Define thread and pthread with their applications.

Ans.

Thread:

Technically, a thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system.

A thread is a semi-process that has its own stack, and executes a given piece of code. Unlike a real process, the thread normally shares its memory with other threads (where as for processes we usually have a different memory area for each one of them). A Thread Group is a set of threads all executing inside the same process. They all share the same memory, and thus can access the same global variables, same heap memory, same set of file descriptors, etc. All these threads execute in parallel (i.e. using time slices, or if the system has several processors, then really in parallel).

Applications:

- One area in which threads can be very helpful is in user-interface programs. These programs are usually centered around a loop of reading user input, processing it, and showing the results of the processing. The processing part may sometimes take a while to complete, and the user is made to wait during this operation. By placing such long operations in a separate thread, while having another thread to read user input, the program can be more responsive. It may allow the user to cancel the operation in the middle.

- In graphical programs the problem is more severe, since the application should always be ready for a message from the windowing system telling it to repaint part of its window. If it's too busy executing some other task, its window will remain blank, which is rather ugly. In such a case, it is a good idea to have one thread handle the message loop of the windowing systm and always ready to get such repain requests (as well as user input). Whenever this thread sees a need to do an operation that might take a long time to complete (say, more then 0.2 seconds in the worse case), it will delegate the job to a separate thread.

- It needs to handle several download requests over a short period, Hence more efficient to create (and destroy) a single thread for each request. Multiple threads can possibly be executing simultaneously on different processors

PThread:

Pthreads is a standardized model for dividing a program into subtasks whose execution can be interleaved or run in parallel. The “P” in Pthreads comes from POSIX (Portable Operating System Interface), the family of IEEE operating system interface standards in which Pthreads is defined (POSIX Section 1003.1c to be exact). There have been and still are a number of other threads models—Mach Threads and NT Threads, for example. Programmers experience Pthreads as a defined set of C language programming types and calls with a set of implied semantics. Vendors usually supply Pthreads implementations in the form of a header file, which you include in your program, and a library, to which you link your program.

Applications:

Numerous systems implement the Pthreads specification; most are UNIX-type systems, including Linux, Mac OS X, and Solaris. Although Windows doesn’t support Pthreads natively, some third- party implementations for Windows are available.

Most hardware vendors now offer Pthreads in addition to their proprietary threads.

Q2. Write a detailed Report on the standards: (In terms of their applications

& pros / cons.)

- **POSIX**
- **OpenMPI**

Abstract:

When increase no longer could be achieved by just increasing the number of transistor on a chip and the clock speed due to high power consumption and heat other means to increase performance had to be developed. One technique that is used to counter the problem is multi-core processors.

Today it's very common with a multi-core processors in most computers, laptops, work stations, servers, phones and tablets. Adding just the cores does not means high performance, in some cases it can actually mean worse performance because the individual cores may not have as high clock speed as a single core processor. It's is absolutely vital that the software running on the hardware utilize the cores that the processor offer in order to get enhanced performance.

Introduction:

Developing software that can take advantage of multiple cores can be done in many different programming languages. If the languages does not have a standard library that supports threads there is often a third party library that can be added to allow usage of threads. For all experiments in this paper C was used as main language, reasoning for using C was that C has been and is a very popular programming language and is used in many different areas of development, GCC compiler on Linux also has support for the two libraries that was tested. The threading techniques that was investigated in this papers was POSIX Threads and OpenMP. Today multithreaded computers are everywhere so it's important that the software utilizes this. Pthreads and OpenMP has been around for some time and are both well tested techniques to parallel computing. But Pthreads tends to be more common than OpenMP.

Background:

POSIX Threads

The POSIX standard states that threads must share a number of things, for example the threads must share:

• Process ID

• Parant Process ID

• Open FD's (File Descriptors)

POSIX Threads offers a header file and a library that must be included at compilation time or else the program wont compile because functions and data structures wont be found by the compiler. POSIX Threads offers a lot of functions to create, manipulate, synchronise and kill threads in a program. Starting point for threads in a Pthreads program is a function called pthreads_create, this function takes a pointer to a function that is the starting point for the new thread. Pthread_exit then is used to kill the thread and in the main thread pthread_join is used to wait for the executing threads.

OpenMP

OpenMP is available for three different programming languages, Fortran, C and C++. It consist of a number of compiler directives that can be used in the source code, these directives need functions to run and those functions is specified in a library that must be included at compile time. It works by letting the compiler translate the directives in to functions that is found in the OpenMP library, it does this at compile time. That means that the compiler must have support for OpenMP, if it does not have support for OpenMP it won't know what to do with the directives. In most cases this won't crash the application, that is a sub goal of OpenMP. It should be portable between different systems, even systems that does not support it. Of course these system won't have the increased performance from parallel computing but they should still be able to run the program.

Empirical Study Findings

POSIX vs OpenMP

OpenMP

The result that OpenMP achieves when put against a sequential version of matrix multiplication is that it does not outperform it in the beginning. On the small matrices the sequential version is faster than OpenMP, first at matrices with a size of 128 x 128 or bigger OpenMP starts to get ahead. After 128 x 128 matrices the gap just keeps growing for each step in OpenMP's favour. Comparing the execution time after 128 x 128 matrices we can clearly see that there is a performance gain from paralleling the algorithm.

PThreads

Pthreads does outperform the sequential version on most of the test. On the smallest matrix it is outperformed by the sequential version and on the 64 x 64 matrix it has the same execution time. After that the Pthread version start to perform better than the sequential version. Then the gap keeps increasing through all the tests, and after 128 x 128 matrices a benefit is gained from paralleling the algorithm

Results

- OpenMP is very capable to help computation intensive algorithms to take advantage of a multicore architecture, in most cases. It does not perform well when working with recursion, it has problem with expressing clear instructions for a recursive problem. It increased the execution speed for two of the three tested algorithms compared to the sequential version.

- Pthreads is very capable to help computation intensive algorithms to take advantage of a multicore architecture. It increased the execution speed for all tested algorithms compared to the sequential version.
- The OpenMP version of Matrix Multiplication and Mandelbrot set calculations outperformed the Pthreads versions. Pthreads outperformed OpenMP on the recursive Quick Sort.
- Using OpenMP resulted in fewer cache-misses than with Pthreads.
- It is not worth paralleling Matrix Multiplication for small input sets, adds unnecessary complexity to the source code and can actually decrease performance.
- Pthreads required the most effort to implement, OpenMP required only small modifications to the source code while Pthreads required a totally tailored source code.

Conclusion

The main focus for the research was the performance offered and effort required to implement a program with Pthreads and OpenMP. The interesting here is that the two models take a very different approach to allow parallel computing, Pthreads takes a more low level approach to threading in the sense that it requires a more tailored program than OpenMP. OpenMP on the other hand is more high level and does not require a very tailored program, it tends to not modify the source code as much as Pthreads does to parallel a piece of software. Another aspect that was investigated were the amount of effort that was required to be able to achieve increased performance using the models.