Henrik Swahn, Software Engineering
hesw91@gmail.com/h_swahn@hotmail.com
11 februari 2016

Pthreads and OpenMP, a comparison

Introduction and Background

During my years at BTH i come in contact with different areas of Software Engineering. The area i find most interesting is parallel computing. In this study i want to explore the area by working with threads. More specifically i intend to take two different threading models available in C++ and performance a series of experiments on them to see how they perform. The two threading models that will be researched is POSIX threads and OpenMP. POSIX threads(Pthreads) is a very low level API to work with threads. It gives a lot of control to the programmer because the programmer has to implement almost everything[1]. At the other end of the spectrum we have OpenMP which represents a high level API to working with threads. It shields a lot more that Pthreads from the programmer by using compiler directives, library routines and environment variables to work with threads[2]. For Software development estimating the development effort is a important part of the process. Estimating wrong can be devastating for a project and the company[5]. To make the right estimation the right decision regarding what techniques to use needs to be done. I want to take a deeper look at the effort required with both the threading API's in contrast to the performance gain they offer.

Target area

The main area of investigation for this study is the different performance that POSIX threads and OpenMP offer. Performance can be many things but in this study the main target is the execution speed that can be achieved with the techniques. To be able to measure the execution speed both the techniques will be used to implement three algorithms. Beside the parallel version of the technique a sequential version of the algorithm will be used as a reference so it's easy to check how much was gained by paralleling the algorithm with the

selected technique. While implementing the algorithms data about the development effort(number of lines) will be collected so i can use this when i look at the results from the algorithms. Then the idea is to look how the threading models performance relative to the effort they require to implement.
The environment is important for these experiments because the POSIX threads that is one of the targets of this study is part of the POSIX C libraries, this means that the experiments will have to be performed on a POSIX system[6]. That is why the environment that will be used for the experiments in this study will be Linux. C++ is the chosen language and it can be used because C++ can take use of C libraries including the POSIX C libraries.

Motive

When the increase in CPU speed was halted because the power consumption and heat became to high some other technique had to be developed in order to increase performance, multi-core processors was a solution to this[3]. So today almost every system uses a multi-core processor or a CPU with threading support. But just adding a multicore processor does not solve all the problem, actually it can be worse since individual cores may not have as high clock speed as just a single core processor[7]. To make a difference the software has to take advantage of the multiple cores on the processor. For writing software that can take advantage of this C++ is one of the most used language on the market and that is why it is very relevant to take a look at this language and the two of the available threading models for it[4]. From a companies point of view it is important to be as efficient as possible and this include choosing the right technique for the system they are going to build or maintain. So it is very important to utilize parallelism to it's best capabilities and that is where i hope to add something.

Value

This is an important topic because many programs are written in C++ and it is important to select a threading model that is appropriate for the goal. To gain understanding of how both the techniques works, how much effort required compared to the performance gain can help programmer make the right choice which in order can improve the development process.

It's also a very interesting to make this comparison because pthread represents a low level approach to threading and openMP represent a high level API to threading. The interesting

here is to see that if we can get similar execution time from two very different approaches, or if we get very different result that can also be very interesting.

News worth

When researching this area i came across some studies that brings up just the execution speed of different models and approaches. What i want to add with my study is to also take the development process into an account when looking att the performance. Because this aspect is not researched in the same extend as just execution speed.

Preliminary Goals

- ~~Find interesting algorithms to perform experiments on~~
- Find a third algorithm to perform experiments on
- ~~Define research questions~~
- Do research into the questions to see if they are realistic
- Gain further understanding how the different models work

Research Questions

1. What kind of earlier work has been done in that does measurements of POSIX threads and OpenMP in C++?
2. How does the OpenMP and POSIX threads work compared to each other?
3. What performance gain is possible with OpenMP and POSIX threads?
4. How does the performance gain differ depending on the input set?
5. How does the two techniques performance compared to each other with the development effort taken into account?

Research Methodology

Literature study

Question one and two is focused around previous work and how the POSIX threads and OpenMP work. This falls under the literature study and will require searching through a lot of previous work. To work as efficient as possible i will have to keep track of my history and what i searched for and what outcome it had. I will use a search schedule to keep track of what i searched for, example:

Date	Location	Query	Findings
20160210	Google Scholar	Pthreads AND Performance	Found useful information about Pthread performance in work: …

The searching will be mainly in google schoolar, summon and IEEE Explore when it comes to information regarding what has been done earlier in the field. When i look into how the techniques work i will also utilize the databases mentioned above but also look at the reference manuals available.

The criteria for the information that i will be looking at will not be limited to research work. When looking at what has been researched earlier in this area i will of course have to look at it but not exclusively when i look at how the techniques work as mentioned earlier. Since techniques evolve all the time it will be important that the papers i read is not to old and refer to very early version of the techniques i will be testing because then the results may very a lot. When looking at the for information in the databases i will be careful about which works i use in my own, i will base the quality of the work somewhat on the number of times it's been referenced in other work.

Empirical study

My empirical study will consist of experiments on POSIX threads and OpenMP. I will develop three different algorithm in both POSIX threads and OpenMP. So far i have two of the algorithms already decided, Dijkstra Algorithm and Quicksort. First a sequential version will be build and then a parallel version will be developed. So when the third algorithm has been decided this mean that 3 * 3 = 9 programs will be developed. Three sequential, one for each algorithm and then two for each parallel version, one in Pthreads and one in OpenMP, will be developed.

For Dijkstras algorithm three experiments will be performed for the three implementations. Each implementation will execute on a small set, a medium set and a very large set to see how the performance look on different input set.

For Quicksort the size of the set will also be tested. So here each implementation will also execute on a small set, a medium set and a very large set.

What will be evaluated is the performance gain that the two techniques offer, this is the reason for developing the sequential version, it gives a reference point to measure the performance gain. The required effort will also be evaluated when developing the parallel version of the algorithm. When have both the performance gain and the effort required a estimation can be performed to decide which technique offers the best performance gain to the cheapest effort.

To be able to do the final estimation and find which technique offer the best performance gain to the cheapest effort data needs to be collected. What is considered performance in this study is the execution time. So when executing the the different versions of the algorithms the interesting data that will be collected is the execution time. But when looking at the effort i was first thinking that implementation time could be interesting to look at but it will not be bias since i have worked a little more with POSIX threads. So instead of implementation time the effort will be measured in the number of lines required to implement a certain level of parallelism. I was thinking something in style with this:

	Sequential	POSIX threads(sec)	Effort(lines)	OpenMP(sec)	Effort(lines)
Quicksort S	10	3	78	2.5	23
Quicksort M	12	4	78	4.3	23
Quicksort L	14	5	78	5	23
……					

References

1. Linux Manual. 2015. PThreads, POSIX Threads Manual. http://man7.org/linux/man-pages/man7/pthreads.7.html
2. OpenMP Architecture Review Board. 2013. OpenMP API Version 4.0. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
3. Balaji Venu. Multi-core processors - An overview. http://arxiv.org/pdf/1110.3535.pdf
4. Tiobe Software. 2016. Top programming language of 2015. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
5. Guidlines for Software Development Effort Estimation, Dirk Basten, Ali Sunyaev, 2011. http://ieeexplore.ieee.org.miman.bib.bth.se/stamp/stamp.jsp?tp=&arnumber=6036100
6. GNU Manual POSIX, 1.2.2 POSIX (The Portable Operating System Interface) http://www.gnu.org/software/libc/manual/html_node/POSIX.html
7. The Free Lunch is Over, A Fundamental turn towards Concurrency in Software, Herb Sutter http://mondrian.die.udec.cl/~mmedina/Clases/ProgPar/Sutter%20-%20The%20Free%20Lunch%20is%20Over.pdf