CSE-4001-Parallel and Distributed computing

Name: O G Ragavi

Reg No: 20BCE1988

Title: Scholarly articles on OpenMP

Lab : 7

Scholarly article 1: Research on OpenMP Model of the Parallel Programming Technology for Homogeneous Multicore DSP

Abstract:

As application complexity continues to grow, using multicore processors has been proved to be an effective methodology to meet the ever-increasing processing demand across the industry association. The Master/Slave model, the Data Flow model and the OpenMP model are the three dominant models for parallel programming. In this paper, the first two models are briefly discussed while the OpenMP model is focused. Some factors (e.g. the number of threads, the scheduling strategy, the load balance, etc.) that affect the execution performance of OpenMP programs were also studied in this paper.

Synopsis:

The use of parallel programming in the IT industry has grown significantly over the past few years.

Using the multi core processors to implement high performance computing has become the consensus of the industry. OpenMP is widely used in general purpose processors for its scalability and flexibility. Different hardware platforms correspond to different parallel programming models. The method using the OpenMP to realize the rapid and efficient image edge detection is presented in this paper. This paper presents a method of taking advantage of the OpenMP model to realize the image edge detection within the platform of TMS320C6678 Digital Signal Processors. The modification of the program only involves the number of cores which proves the Open MP model has a good scalability.

Introduction:

- Due to the limitation of the CMOS technology, it is difficult to continuously improve the frequency of the processor chip after it reaches 4GHz. Moore's Law leading the semiconductor market nearly 40 years may fail in the next 10 to 20 years. Using the multi core processors to implement high performance computing has become the consensus of the industry. Multicore Digital Signal Processors (DSP) are widely used in wireless telecommunication, industrial control, audio and video processing, etc. As embedded multicore hardware enables more functions to be implemented on the same device, efficient parallel programming methods are required to achieve desired performance without increasing software complexity.
- OpenMP is now widely used in general purpose processors for its scalability and flexibility, while the application in multicore DSP is still in the initial stage. The state-of-the-art approach to program multi-core DSPs is based on proprietary vendor Software Development Kits(SDKs), which only provides low-level, non-portable primitives
- Different hardware platforms correspond to different parallel programming models. This paper will take this platform as an instance, analyze the number of threads, the scheduling strategy on the performance of the OpenMP model. The method using the OpenMP to realize the rapid and efficient image edge detection is also presented in this paper.

Highlights:

- This paper presents a method of taking advantage of the OpenMP model to realize the image edge detection within the platform of TMS320C6678 Digital Signal Processors (DSP)
- We study the two main factors which will influence the performance of OpenMP model

- The best performance can be obtained at the point where the number of threads is equal to the number of cores which are available within the platform
- In terms of image edge detection, we adopt the runtime scheduling strategy which gets a better performance compared to others
- The modification of the program only involves the number of cores which proves the OpenMP model has a good scalability.

Study subjects and analysis:

The experimental results show that the OpenMP model has a better advantage on scalability and flexibility compared to the Master/Slave model and the Data Flow model.The Data Flow model is used for distributed control and execution

Results:

TABLE I. EXECUTION TIME VERSUS NUMBER OF THREADS

	Number of threads		
	4	8	12
Execution Time/cycle	17112903	10353808	16902841

The table shows the text result that execution time varies with the number within the DSP platform. As can be seen, the execution time will bottom at a point where the number of threads is equal to the number of cores which are available within the platform. If the number of threads is less than the number of cores, then there will be some cores in idle state. Inversely, too many threads will result in the increasing overhead of threads creating and switching. Both conditions above are not conducive to performance improvement.

- When the SIZE is 80, then there are 80x80 multiply-add operations, the execution time is 19.2ms when the schedule(static) is specified. Since the main objective is to compare the pros and cons between various scheduling mechanisms, we want to calculate the relative value among them.

Conclusion:
OpenMP is one of the most widely used parallel programming techniques in the modern multicore era. But applying this model to homogeneous multicore DSP is still a great challenge. In this paper, we studied the two main factors which will influence the performance of the OpenMP model. The best performance can be obtained at the point where the number of threads is equal to the number of cores which are available within the platform. The scheduling strategy is determined by the specific application. In terms of image edge detection, we adopt the runtime scheduling strategy which gets a better performance compared to others. For the sake of validating the speedup, we use 1 to 8 cores to realize the edge detection and record the respective cost. The modification of the program only involves the number of cores which proves the OpenMP model has a good scalability. **The conclusion that the speedup increases linearly with the number of cores satisfies Gustafson's law. What's more, under the case of 8 cores running simultaneously, the speedup reaches 7.233.** Compared with the Master/Slave model, the OpenMP model has approximately improved one-third on performance.

Scholarly article2: The Design of OpenMP Tasks

Abstract:

OpenMP has been very successful in exploiting structured parallelism in applications. With increasing application complexity, there is a growing need for addressing irregular parallelism in the presence of complicated control structures. This is evident in various efforts by the industry and research communities to provide a solution to this challenging problem. One of the primary goals of OpenMP was to define a standard dialect to express and to exploit unstructured parallelism efficiently.This paper presents the design goals and key features of the tasking model, including

a rich set of examples and an in-depth discussion of the rationale behind various design choices. It compares a prototype implementation of the tasking model with existing models, and evaluates it on a wide range of applications. The comparison shows that the OpenMP tasking model provides expressiveness, flexibility, and huge potential for performance and scalability

Introduction:

In the last few decades, OpenMP has emerged as the de facto standard for shared-memory parallel programming. OpenMP provides a simple and flexible interface for developing portable and scalable parallel applications. OpenMP grew in the 1990s out of the need to standardize the different vendor specific directives related to parallelism. It was structured around parallel loops and was meant to handle dense numerical applications.

The set of features in the OpenMP 2.5 specification is ill equipped to exploit the concurrency available in modern applications.Users now need a simple way to identify independent units of work and not concern themselves with scheduling these work units. This model is typically called “tasking”.With the goal of defining a simple tasking dialect for expressing irregular and unstructured parallelism, a subcommittee of the OpenMP 3.0 language committee was formed.

Nested Parallelism Vs Tasking:

New OpenMP tasks allow a programmer to express parallelism that in OpenMP 2.5 would be expressed using nested parallelism. The versions using nested OpenMP, while simple to write, usually do not perform well because of a variety of problems (load imbalance, synchronization overheads etc.).Handling recursive code structures is another simple case that motivated tasking in OpenMP was recursive work generation.Nested parallelism can be used to allow recursive work generation but at the expense of the overhead in creating a rigid tree structure of thread teams and their associated (unnecessary) implicit barriers.

Prototype implementation:

In order to test the proposal in terms of expressiveness and performance, they have developed their own implementation of the proposed tasking model . They developed the prototype on top of a research OpenMP compiler (source-to-source restructuring tool) and runtime infrastructure. The runtime infrastructure is an implementation of a user level thread package based on the nano-threads programming model introduced first by Polychronopoulos.

The nano-thread layer is implemented on top of POSIX Threads (also known as pthreads).They decided to use pthreads to ensure that they will be portable across a wide range of systems. This layered implementation can have a slight impact on efficiency.

Once the task is first executed by a thread, and if the task has task scheduling points, we can expect two different behaviors. First, the task is bound to that thread (so, it can only be executed by that thread), and second, the task is not attached to any thread and can be executed by any other thread of the team. The library offers the possibility to move a task from the team queues to the local queues. This ability covers the requirements of the united clause of the task construct, which allows a task suspended by one thread to be resumed by a different one. The synchronization construct is provided through task counters that keep track of the number of tasks that are created in the current scope (i.e., the current task). Each task data structure has a successor field that points to the counter the task must decrement.

Conclusion:

They have presented the work of the OpenMP 3.0 tasking subcommittee which was a proposal **to integrate task parallelism into the OpenMP specification**. This proposal allows programmers to **parallelize program structures like while loops and recursive functions more easily and efficiently**. We have shown that, in fact, these structures are easy to parallelize with the new proposal.

The process of defining the proposal has not been without difficult decisions, as they tried to achieve conflicting goals: simplicity of use, simplicity of specification, and consistency with the rest of OpenMP.Their discussions identified trade-offs between the goals, and our decisions reflected our best judgments of the relative merits of each.
The comparisons of these results show that expressiveness is not incompatible with performance and the OpenMP tasks implementation can achieve very promising speedups when compared to other established models. Overall, **OpenMP tasks provide a balanced, flexible, and very expressive dialect for expressing unstructured parallelism** in OpenMP programs.