IMPLEMENTATION OF TASK PARALLELISM USING PYTHON-JUG

Arshiya A. Sheikh
Master of Engineering (M.E.) Scholar
Department of Computer Science & Engineering,
Sant Gadge Baba Amravati University, Amravati, India

Abstract : In this paper we proposed Architecture to Implement a Task Parallelism using Jug: A Task-Based Parallelization Framework. The paper presents the efficient way to slice the program depending on the task and running the program parallel.

***Keywords* – Parallel Computing, Program Slicing**

1. Introduction:

Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. [1] As power consumption (and consequently heat generation) by computers has become a concern in recent years,[2] parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors. [3] Parallel computers can be roughly classified according to the level at which the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters, MPPs, and grids use multiple computers to work on the same task. Specialized parallel computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks. Parallel computer programs are more difficult to write than sequential ones, because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are typically some of the greatest obstacles to getting good parallel program performance. The maximum possible speed-up of a single program as a result of parallelization is known as Amdahl's law.

2. What is jug?

Jug is a simple way to write easily parallelisable programs in Python. It also handles intermediate results for you. The main unit of jug is a Task. Any function can be used to generate a Task. A Task can depend on the results of other Tasks. The original idea for jug was a Makefile-like environment for declaring Tasks. I have moved beyond that, but it might help you think about what Tasks are. You create a Task by giving it a function which performs the work and its arguments. The arguments can be either literal values or other tasks (in which case, the function will be called with the result of those tasks!). Jug also understands lists of tasks and dictionaries with tasks

Task Generators In the code above, there is a lot of code of the form Task(function,args), so maybe it should read function(args). A simple helper function aids this process:

```
from jug import TaskGenerator
computefeatures = TaskGenerator(computefeatures)
kmeans = TaskGenerator(kmeans)
compute_bic = TaskGenerator(compute_bic)
@TaskGenerator
def Nr_Clusters(bics):
return argmin(bics)
```

You can obtain the final results of your computation by setting up a task that saves them to disk and loading them from there. If the results of your computation are simple enough, this might be the simplest way. Another way, which is also the way to access the intermediate results if you want them, is to run the jug script and then access the result property of the Task object

3. Implementation:

The Task parallelism is achieved by Jug. Jug allows you to write code that is broken up into tasks and run different tasks on different processors. It currently has two backends. The first uses the filesystem to communicate between processes and works correctly over NFS, so you can coordinate processes on different machines. The second is based on redis so the processes only need the capability to connect to a common redis server.

Jug also takes care of saving all the intermediate results to the backend in a way that allows them to be retrieved later.

Task parallelism (also known as function parallelism and control parallelism) is a form of parallelization of computer code across multiple processors in parallel computing environments. Task parallelism focuses on distributing execution processes (threads) across different parallel computing nodes. It contrasts to data parallelism as another form of parallelism.

In a multiprocessor system, task parallelism is achieved when each processor executes a different thread (or process) on the same or different data. The threads may execute the same or different code. In the general case, different execution threads communicate with one another as they work. Communication usually takes place by passing data from one thread to the next as part of a workflow.

The pseudocode below illustrates task parallelism:

```
program:
...
if CPU="a" then
do task "A"
else if CPU="b" then
do task "B"
end if
...
end program
```

Figure: Pseudocode of task parallelism

The goal of the program is to do some net total task ("A+B"). If we write the code as above and launch it on a 2-processor system, then the runtime environment will execute it as follows.

- In an SPMD system, both CPUs will execute the code.
- In a parallel environment, both will have access to the same data.
- The "if" clause differentiates between the CPU's. CPU "a" will read true on the "if" and CPU "b" will read true on the "else if", thus having their own task.
- Now, both CPU's execute separate code blocks simultaneously, performing different tasks simultaneously.

Code executed by Client "A"

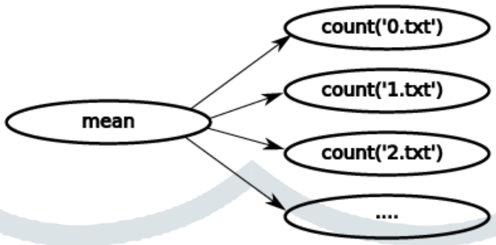
```
program:
...
do task "B"
...
end program
```

Figure : Code executed by Client "A"

```
program:
...
do task "A"
...
end program
```

Figure : Code executed by Client "B"

As a simple example, if we are running code on a 2-processor system (CPUs "a" & "b") in a parallel environment and we wish to do tasks "A" and "B", it is possible to tell CPU "a" to do task "A" and CPU "b" to do task 'B" simultaneously, thereby reducing the run time of the execution. The tasks can be assigned using conditional statements as described below. Task parallelism emphasizes the distributed (parallelized) nature of the processing (i.e. threads), as opposed to the data (data parallelism). Most real programs fall somewhere on a continuum between task parallelism and data parallelism.



4. Conclusions:

We Implementation a Novel method for Task parallelism using Jug. The Key idea of this project is to slice the program in Number of slice using Jug so that the individual slice can run independently.

5. References:

[1] S.V. Adve et al. (November 2008). "Parallel Computing Research at Illinois: The UPCRC Agenda" (http://www.upcrc.illinois.edu/ documents/UPCRC_Whitepaper.pdf) (PDF).

[2] Asanovic et al. Old [conventional wisdom]: Power is free, but transistors are expensive. New [conventional wisdom] is [that] power is expensive, but transistors are "free".

[3] Asanovic, Krste et al. (December 18, 2006). "The Landscape of Parallel Computing Research: A View from Berkeley" (http://www.eecs. berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf) (PDF). University of California, Berkeley. Technical Report No. UCB/EECS-2006-183.