3 Parallel Paradigms

There are 3 main type of paradigms to consider when designing parallel programs. They are master-worker, result-oriented and pipeline-oriented. To find more information about parallel programming refer to this paper http://www.lindaspaces.com/book/book.pdf Below are descriptions of each type of paradigm as well as an analogy of how building a house would be done and sudo code to use each paradigm to calculate the dot product. The house building analogies deal with different ways of organizing the workers (threads) to build a house. The following is the c++ code for calculating the dot product sequentially:

```
float dot_product(float *a,float *b,int size)

{

   float dp = 0.0f;

   for (int i=0;i<size;i++)

      dp += a[i] * b[i];

   return dp;

}
```

Master-Worker (agenda or blackboard)

The master-worker paradigm works by having one thread be the master thread and delegate tasks to the other threads (workers). The workers may have to ask the master thread for more work repeatedly. The master thread may also hold an "agenda" of tasks and allow threads to pick tasks until the job is completed.

House Building Analogy:

"The tasks on the agenda might occur in a single unordered collection(we tell the workers "grab any task from this list"), or there might be a sequence ("first do all these tasks, then start on those"). We need a foundation (foundation building might be represented by a collection of tasks on the agenda); then we need a frame; then we need a roof; then we need wallboard and perhaps plastering, and so on. We assemble a work team of generalists, each member capable of performing any construction step. First, everyone pitches in and builds the foundation; then, the same group sets to work on the framing; then they build the roof; then some of them work on plumbing while others (randomly chosen) do the wiring, and so on."[1]

[1] N. Carriero and D. Gelernter, HOW TO WRITE PARALLEL PROGRAMS A FIRST COURSE (MIT, 1990) 13.

Pseudo Code:

```
dot_product(a,b,size, workerNum)

    initilize variables and threads

    the master assigns calculations for different values of i to threads

    when a worker finishes ask the master for another computation

    collect data from workers

    cleanup threads
```

Result–Oriented (dataflow)

The idea behind the result oriented paradigms is to start by designing the finished product. Then divide the finished product into separate components. Each thread works on a separate piece of the overall product.

House Building Analogy:

“The result (the house) can be divided into many separate components—front, rear and side walls, interior walls, foundation, roof and so on. After breaking the result into components, we might proceed to build all components simultaneously, assembling them as they are completed; we assign one worker to the foundation, one to the front exterior wall, one to each side wall and so on. All workers start simultaneously. Separate workers set to work laying the foundation, framing each exterior wall, building a roof assembly. They all proceed in parallel, up to the point where work on one component can't proceed until another is finished.”[2]

Pseudo Code:

```
dot_product(a,b,size, workerNum)

    initilize variables and threads

    divide the range from 0 to size into even chunks

    give each worker process a chunk

    cleanup threads
```

[2] N. Carriero and D. Gelernter 12.

Pipeline-Oriented (specialist)

The pipeline-oriented paradigm works by having a group of threads each work on different parts of multiple copies of the same problem at the same time. The partially completed problems then get passed to a different group of threads like an assembly line until the set of problems are finished.

House Building Analogy:

“we might envision parallelism by starting with the crew of workers who will do the building. We note that house-building requires a collection of separate skills: we need surveyors, excavators, foundation-builders, carpenters, roofers and so on. We assemble a construction crew in which each skill is represented by a separate specialist worker. They all start simultaneously, but initially most workers will have to wait around. Once the project is well underway, however, many skills (hence many workers) will be called into play simultaneously: the carpenter (building forms) and the foundation-builders work together and concurrently, the roofer can be shingling while the plumber is installing fixtures and the electrician is wiring, and so on. This approach is particularly suited to pipelined jobs—jobs that require the production or transformation of a series of identical objects. If we're building a group of houses, carpenters can work on one house while foundation-builders work on a second and surveyors on a third.”[3]

Pseudo Code:

```
dot_product(a,b,size, workerNum)

    initialize variables and threads

    assign each thread a different value of i to do the dot product calculation for

    as the first thread completes its calculation reassign it to the next value of I that
        does not have a thread allocated to it

    cleanup threads
```

[3] N. Carriero and D. Gelernter 12.