Title

Program Execution on Multiple Devices

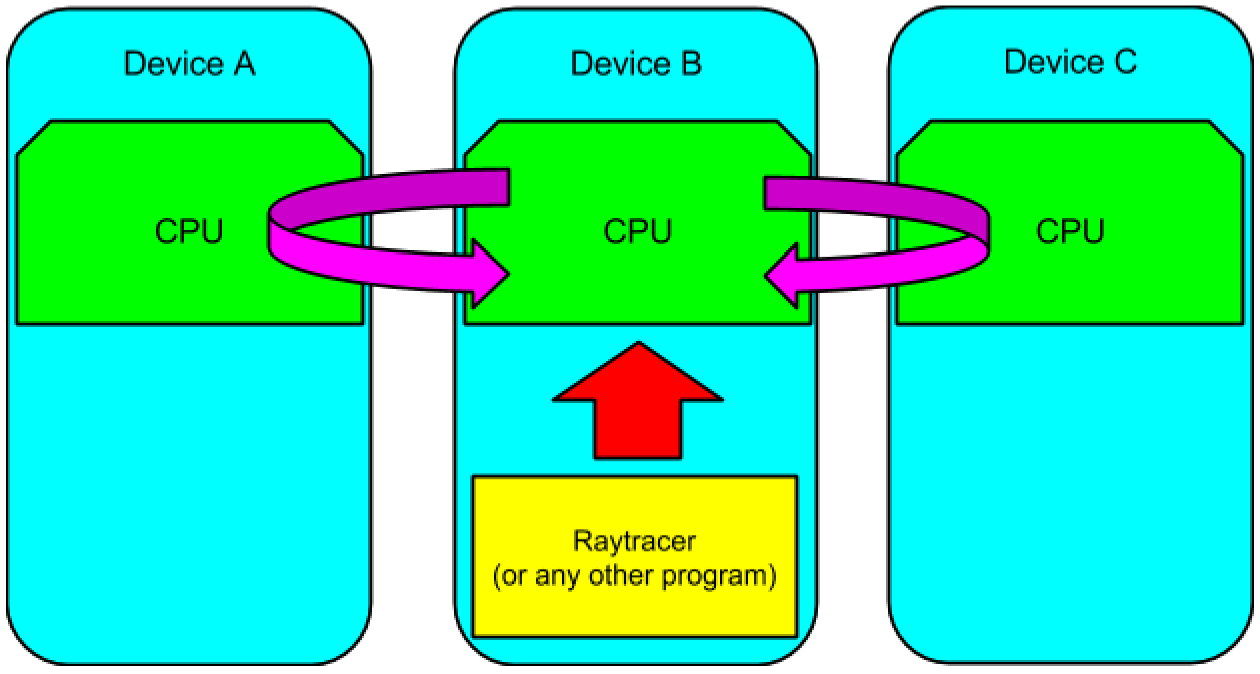
Summary

We are going to implement a library interface and an environment that allows a program to complete its execution on multiple devices. We want to construct a network of devices that share their computation powers among each other.

Background

Normally speaking, sending binary/source code to other devices to execute each new job is inefficient. Since functions usually have high reusability rates, we decide to implement an interface for developers to set up a running environment on other devices beforehand such that they upload whatever binary/source code only once. Then, when there are jobs to be executed, the local device can treat other devices as “servers”, and only send requests of function arguments. We believe that the latency of network communication is tolerable because some online gaming is done in a similar way, and so our interface should be applicable to many applications.

We plan to use a ray-tracer to demonstrate the how our library interface works and the actual speedup the program can achieve. Hence, we are going to use our library interface to write a parallel ray-tracer, instead of using any existing parallel libraries like OpenMP. This allows our library and the running environment handle how the jobs are distributed and executed.



The Challenge

1. Choosing the most generalizable way for jobs to be distributed and executed such that developers are not limited to what specific program they implement, which platform to run on, etc.
2. Carefully considering time constraints of some jobs, network communication latency, device computational power, possibile crashes, etc.
3. Sharing the state of a worker server to other servers (information about workload of a specific request increases and worker server crashes).
4. Making sure the security of the running device is not compromised.

Resources

1. Platforms: experimenting on all popular platforms: Linux, Windows, Mac.
2. Paper on real time dynamic scheduling algorithms with fault tolerance.
3. Resources on running bytecode cross-platform: For now we decide to use JVM.

Deliverables

1. Plan to achieve:
    a. Implement the environment that's able to:
        i. Load and run JVM bytecodes received from other devices.
        ii. Compile and send binary from current device to other devices to run.
        iii. Schedule jobs according to network latency, CPU power, etc.
    b. Implement library interface such that developer can create a job and distribute it to workers. Implement a parallel ray-tracer using the library interface developed, and measure speedup with at least 4 devices.
2. Hope to achieve:
    a. Make the environment and library interface handle deadline, fallback policy, etc.
    b. Port the project to smartphones.

Platform Choice

Hardware: Cross-platform. Any device that's able to run JVM.
Software: JVM, because its bytecode is cross-platform, and languages other than Java can also be compiled into Java bytecode.

Schedule

Week 1

JVM execute task

Week 2

Simple task scheduling

Library interface

Week 3

Send/receive tasks

Ray-tracer demo

Week 4

More sophisticated task scheduling (with network latency, bandwidth, CPU)

Ray-tracer demo

Week 5

More sophisticated task scheduling (with network latency, bandwidth, CPU)
Ray-tracer demo

Week 6

Ray-tracer demo