Implementation and Evaluation of MPI on STM32

https://coder6583.github.io/15418-project-page/

Soma Narita, Josef Macera
URL:

Summary

We will implement a minimal MPI-like message passing library on STM32 microcontrollers by extending our custom RTOS that we wrote in 18-349. We will evaluate its scalability by testing different parallel workloads, and by building a real-time sound source localization application running on multiple STM32s, each with a microphone collecting audio data.

Background

We plan to build a sound source detection system on four STM32 microcontrollers, each connected to a microphone collecting audio data. The digital signal processing will be divided and parallelized on the microcontrollers, and the result will be gathered to a master STM32.

Below is the system flow:

1. Each STM32 collects audio data from a microphone
2. The sampled audio data will be divided into N intervals, each distributed to one node in the network
3. The data will be sent out using our MPI implementation
4. Each board will perform digital signal processing to generate a sound heat map on its assigned partition. This will include FFT calculations, prefiltering/anti-aliasing, time delay estimation, cross-correlation calculations, and solving 2D geometric problems to go from time delay to physical mappings.
5. Partial results will be gathered to a master STM32 that will display the heat map on a display

Challenge

1. Unlike traditional MPI systems, embedded systems have a lot of limitations on their communication bandwidth (UART: around 1Mbps, SPI: around 25Mbps). This introduces a lot of overhead for synchronization, making the system difficult to scale linearly.
2. Embedded systems also have a small RAM (around 96KB), severely limiting the work that can be assigned to a STM32 at a time. We must carefully tune our sampling frequency in order to find a happy medium between communication frequency and task granularity.
3. Embedded real-time systems also can have strict timing requirements - if our computation code is inefficient, then our threads might not meet scheduling requirements - because the bulk of the work here is done on signal processing, we must optimize our

code to be as fast and efficient as possible, potentially implementing research papers or approximating solutions.

Through this project, we hope to learn the tradeoff between communication and computation on limited hardware. We aim to evaluate if using MPI on embedded systems will bring similar linear scaling we saw on the PSC/GHC machines with OpenMPI.

Resources

We will use a small network of STM32 microcontrollers as the main compute platform, with one microphone attached to each STM32 and a master STM32 responsible for collecting results and drawing the final sound-intensity heat map. The main code base will not be from scratch: we plan to start from our basic RTOS kernel developed as part of 18-349, and extend it to support an MPI-like library, implemented using UART as the communication layer. We also intend to explore other communication platforms, such as SPI, if bandwidth becomes a bottleneck due to scaling with more processors. We will also reuse basic DSP and matrix-multiply style test programs to validate correctness and measure performance before moving to the sound-localization application.

As references, we expect to use the official MPI 4.1 standard as the main model for message-passing semantics, even though our implementation will only support a very small embedded subset. We will also rely on STMicroelectronics documentation for the specific STM32 board and peripherals we use, especially for UART/SPI, ADC or digital microphone interfaces, timers, and memory/performance limits. ST's documentation for STM32 families and audio acquisition examples should be useful when bringing up the microphone and sampling pipeline.

The main resources we still need to pin down are the exact microphone hardware, the best interconnect choice between boards, and the synchronization method needed for accurate source localization. In particular, triangulation works much better if the nodes are time-aligned, so we may need a shared trigger, timestamping support, or a calibration procedure. We would also benefit from access to oscilloscopes, logic analyzers, and possibly better STM32 development boards with faster serial links or better audio support. To study scaling at larger scales with this model, it is not financially feasible to implement this physically with say, 32 STM32s and 32 mics, due to cost. Thus, we would benefit from using the GHC machines or the PSC machines to simulate the product; to do so, we would then use the standard MPI library, but add timing restrictions (i.e, only be able to send messages every T ms) to reflect bandwidth limitations of embedded systems like UART/SPI/etc.

As a starting point, we intend to read these papers and implement basic versions of it first:
https://ieeexplore.ieee.org/document/1162830
https://www.sciencedirect.com/science/article/abs/pii/S0885230896900248

Goals and Deliverables

We split our goals/milestones into two categories: Plan to achieve, and hope to achieve.
We *plan* to achieve:

- A working implementation of a lightweight MPI library. We plan to implement the functions below, and add more if needed.
    - MPI_Init
    - MPI_Send
    - MPI_Recv
    - MPI_Barrier
    - MPI_Allgather
    - rank/communicator support
- Functional matrix multiplier using multiple STM32s connected with our MPI library
    - Explore how scalable our system is
- Functional sound source detection under loose conditions (low sampling speed, low number of nodes) with acceptable accuracy on four physical STM32/mic nodes. We believe we can achieve this because the parallel workload is minimally distributed and we believe we will find few challenges to get a basic working physical implementation ready. We plan for this to be as a demo at the poster session, as described above - ideally on a flat table with the microphones on the edge of the table and the display in the center. People can walk around the table, clap their hands, talk, and the heat map will update accordingly and in real time.
- As part of the analysis of the scaling of our project, we hope to optimize and design our system to be able to scale linearly in two ways:
    - Fix the sampling frequency of each node, and vary the number of nodes in the system that communicate
    - Fix the number of nodes in the system, and vary the sampling frequency
    - We hope to collect sufficient data and create a sufficient analysis to explain how these limitations impact our ability to scale this system in both directions.

If all goes well, we *hope* to achieve:

- A physical implementation of our system on 8 physical STM32/mic nodes with much, much higher sampling frequency. The specific frequency is not yet determined by us, but we would hope to support linear scaling up to ~100x speedup in frequency, that is, varying f=1kHz to f=100kHz, for example.

Platform Choice

The hardware limitations of the STM32 are a perfect choice for our project, in many ways like communication bandwidth, memory limitations, etc. - Also, given that each STM32 has its own memory, and that there is no “global” memory shared by all nodes, the message passing interface is perfectly suitable to work here. Our kernel is written in C, and it makes sense to continue working on our kernel in C (although writing ARM to support more system calls on our kernel could definitely be a possibility).

Schedule

Week 1

1. Implement SPI syscalls
2. Design MPI API

Week 2

1. Implement MPI_Init, MPI_Send, MPI_Recv
2. Implement rank and communicator
3. Debug

Week 3

1. Implement MPI_Barrier, MPI_Allgather
2. Run matrix multiply

(Milestone)

Week 4

1. Implement audio sampling / Implement display interface
2. Implement basic sound source detection
3. Debug

Week 5

1. Run experiments!
2. Optimize
3. Do report