Petascale Emulation

Joshua Nasman
Adviser: Chris Carothers
Co-adviser: Ken Jansen
Overview

- Motivation
  - PHASTA
  - Larger Computers
  - Goal of Petascale hero runs

- Phases
  - Implementation details of phase 1
  - Overview of future work

- Questions
PHASTA

- Parallel Hierarchic Adaptive Stabilized Transient Analysis
- Phasta can model compressible or incompressible, laminar or turbulent, steady or unsteady flows in 3D, using unstructured grids. [https://www.scorec.rpi.edu/wiki/PHASTA](https://www.scorec.rpi.edu/wiki/PHASTA)
- Goal is to get it to scale to petaflops
Goal: enable accurate prediction of PHASTA performance on larger scale systems than currently available

Other advantages along the way:
- See what part of communication is dominating
- Compare communication vs. computation
- Study the number of MPI calls
Stepping stones

- One instance of PHASTA per processor. Simply use a lookup table. (Detailed in future slides)
- Multiple instances of PHASTA per processor. (Using threading)
- Fully model network
  - Model cache performance as well
Phasta Emulator: 2 modes

Mode 1
Data collection

Mode 2
Emulation
Lookup mode advantages

- Network is not tied to a specific system.
- Simpler to code
- Should provide good results for lightly loaded network
- Can get estimates on performance without the full details of the system
Disadvantages

- On a fully loaded system messages may be greatly delayed
  - (Even on the ‘non-blocking’ network on TACC) we are already seeing issues with the all reduce
Method 1: sending a message
Mode 1: Timing

Address 1: Send

Address 2: Receive

Address 3: AllReduce

Computation

Computation

Computation
Part of the goal of the simulator is to ultimately make it portable.
- Not directly modify the base code
- Use timers which are available across systems
  - MPI_Wtime()

All MPI calls simply call the PMPI equivalent.

By linking against our own MPI calls we can intercept these calls
int MPI_Waitall(int a, MPI_Request * b, MPI_Status * c)
{
    return PMPI_Waitall(a, b, c);
}
All that timing can be handled from within our MPI calls

```c
int MPI_Waitall(int a, MPI_Request * b, MPI_Status * c)
{
    getTimeAndReset();
    retVal = PMPI_Waitall(a, b, c);
    getMPITime(WAITALL);
    return retVal;
}
```

(WAITALL is part of an enumeration)
```c
int getMPItime(int type)
{
    finish = MPI_Wtime();
    mpicount[type]++;
    ourtime = (finish - start);
    MPItotaltime += ourtime;
    if (MPIhead[type] == NULL)
        else
        {
            MPIcurrent[type]->next = malloc(sizeof(timing_data));
            MPIcurrent[type] = MPIcurrent[type]->next;
            MPIcurrent[type]->time = ourtime;
            MPIcurrent[type]->next = NULL;
        }
    start = finish;
    return returnVal;
}

return 0;
```
int getTimeAndReset()
{
    finish = MPI_Wtime();
    ourtime = (finish - start);
    totaltime += ourtime;
    if (head == NULL)
    {
        current = malloc(sizeof(timing_data));
        current = current->next;
        current->time = ourtime;
        current->next = NULL;
    }
    start = finish;
    return returnVal;

    return 0;
}
int inSendList( int toFind)
{
    int_list * iterator = isendlist;
    int found=0;
    while(found==0&&iterator!=NULL)
    {
        if(iterator->data==toFind)
        {
            return 1;
            iterator=iterator->next;
        }
    }
    iterator=isendlist;
    isendlist=malloc(sizeof(int_list));
    isendlist->data=toFind;
    isendlist->next=iterator;
    return 0;
}

//Time will be able to be taken care of in this (slightly modified) call if the array is slightly modified and the time // is returned instead of simply a 1.
The times for the communication is looked up according to data (if system is available) or information about network.

The computation part is multiplied by the speed of the processor.
Future Work

- When the simulator reaches a stage where it’s simulating more processors than on the available system, the emulators will take a more dominant role.
- Nodes will be designated for the simulator. These will be responsible for keeping all the PHASTA instances at the same GVT.
Assumptions

- These implementations assume the following worlds have already been created
  - PHASTA_World: contains all processors running phasta
  - EM_World: contains all processors running emulators
  - PHASTA_Node_World: contains all phasta instances on a given node
Getting the All Reduce to the Simulator

- MPI all reduce calls a `pmapi` send with all the information stored in a buffer so that the simulator can then call it.
- Simulator then calls `pmapi` reduce on PHASTA world and adjust time based on the size of `PHASTA_NODE_WORLD` and how long that many messages should take.
  - Phasta world could also be used for simple case
- In case with many nodes after a simulator calls for its own node it can call an all reduce across `SIM_WORLD`.
- Time would then have to be adjusted again (or only at this step and not at the other)
Continue for how many phasta instances per process. Having fewer on certain nodes should not affect result but should not be taken into account in timing.
All Reduce over nodes
Questions?