HIGH PERFORMANCE COMPUTING CENTER

HPCC

TEXAS TECH UNIVERSITY

Master Slave Paradigm

- What is it?
- What problems fit this paradigm?
- What problems don’t fit?

What is MSP?

- In the context of needing to complete many tasks.
- One processor issues tasks to a single processor in a team. The master processor is responsible for coordination of the entire computation. The slaves perform individual tasks.

What problems fit the Paradigm?

- In general, the best fit for this type of computation involves the need to complete many (perhaps it is unknown how many) tasks.
- Each task may have an unknown run time. This could be due to the nature of the task or the nature of the processors.

Examples

- Adaptive Quadrature
- Global Optimization
- Solving multiple problems, each solve requires an unknown number of iterations. (EX. Root finding, solving ODEs, etc.

Problems not suitable for MSP?

- Problems with regular predictable compute loads on homogeneous systems.

Master/Slave Code

```
/* code based on that of Fikret Ercal U of Missouri
   */

#include <stdio.h>
#include <mpi.h>
#define WORKTAG                 1
#define DIETAG          2
#define MWORK           200
double  A[1000],B[1000],C[1000],D[1000];
void master();
void slave();
```

```
main(argc, argv)
int                 argc;
char                *argv[];
{
    int myrank,i;
    for (i=0;i <1000; i++)
    {
    A[i]=i;
    B[i]=i;
    C[i]=0;
    }
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    if (myrank == 0) {
       master();
         for (i=1;i < MWORK; i++)
            printf ("results C[%d]=%3.0f on %3.0f\n",i, C[i],D[i]);
      } else {
      slave();
      }
      MPI_Finalize();                    /* cleanup MPI */
}
```

```
void master()
{
        int                 com_size, rank, work;
        double              result;
        MPI_Status          status;
        MPI_Comm_size(MPI_COMM_WORLD,&com_size);
        for (rank = 1; rank < com_size; ++rank)
      {
                work = rank;
                MPI_Send(&work,                         /* message buffer */
                        1,                      /* one data item */
                        MPI_INT,                /* data item is an integer */
                        rank,                   /* destination process rank */
                        WORKTAG,                /* user chosen message tag */
                        MPI_COMM_WORLD);/* always use this */
        }
  work++;
```

```
while(work < MWORK){
            MPI_Recv(&result,        /* message buffer */
                  1,                 /* one data item */
                  MPI_DOUBLE,        /* data item is a double real */
                  MPI_ANY_SOURCE,          /* receive from any sender */
                  MPI_ANY_TAG,       /* receive any type of message */
                  MPI_COMM_WORLD,          /* always use this */
                  &status);          /* info about received message */

            C[status.MPI_TAG]=result;
            D[status.MPI_TAG]=status.MPI_SOURCE;

            MPI_Send(&work, 1, MPI_INT, status.MPI_SOURCE,
                        WORKTAG, MPI_COMM_WORLD);
      work++;
     }
     for (rank = 1; rank < com_size; ++rank) {
          MPI_Recv(&result, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
           C[status.MPI_TAG]=result;
           D[status.MPI_TAG]=status.MPI_SOURCE;
}
     for (rank = 1; rank < com_size; ++rank) {
            MPI_Send(0, 0, MPI_INT, rank, DIETAG, MPI_COMM_WORLD);
     }
}
```

```
void slave()

{
        double          result, x=0.;
        int             work,  i, j;
        MPI_Status      status;
      for(;;){
        MPI_Recv(&work, 1, MPI_INT,
0,MPI_ANY_TAG,MPI_COMM_WORLD,&status);
           if (status.MPI_TAG == DIETAG)
                {
           return;
           }
                result = A[work]+B[work];

           /*  begin useless computation to increase time */
                   for(i=0; i<999; i++){
                          for(j= 0; j< 999; j++){
                                 x += 1/(1+i+j);}}
           /*  End useless computation  */

                MPI_Send(&result, 1, MPI_DOUBLE, 0, work,
MPI_COMM_WORLD);
}
}
```

HIGH PERFORMANCE COMPUTING CENTER

HPCC

TEXAS TECH UNIVERSITY