```
------------------------------------------------------------------------------
sssp.flcn
------------------------------------------------------------------------------
int changed=0, hchanged=0;
relaxgraph( Point p,Graph graph){
p.updated=false;
p.updated1=false;
foreach(t In p.outnbrs)
      MIN(t.dist,p.dist+graph.getweight(t,p),changed);
}
reset( Point t,Graph graph){
t.dist=1234567890;
t.olddist=1234567890;
t.updated=false;
}
reset1( Point t,Graph graph){
if(t.dist<t.olddist)t.updated=true;
t.olddist=t.dist;
}
void SSSP(char *name) {
Graph hgraph;
hgraph.addPointProperty(dist,int);
hgraph.addPointProperty(updated,bool);
hgraph.addPointProperty(olddist,int);
hgraph.read(name);
foreach(t In hgraph.points)reset(t,hgraph);
hgraph.points[0].updated=true;
hgraph.points[0].dist=0;
while(1){
changed=0;
foreach(t In hgraph.points)(t.updated==true)relaxgraph(t,hgraph);
if(changed==0) break;
foreach(t In hgraph.points)reset1(t,hgraph);
}
int max=0;
for(int i=0;i<hgraph.npoints;i++){
if(max <hgraph.points[i].dist)max=<hgraph.points[i].dist;
}
printf("%d\n",max);
return;
}
int main(int argc,char *argv[]){
SSSP(argv[1]);
}
```

```
--------------------------------------------------------------------------------
----------------
sssp.h
--------------------------------------------------------------------------------
----------------
#include "DHGraph.h"
#include "DGGraph.cu"
#include "thrust.cu"
#include<sys/time.h>
#include </usr/local/cuda/include/cuda.h>

#include </usr/local/cuda/include/cuda_runtime_api.h>
#include<unistd.h>
#include</usr/local/cuda-5.0/samples/0_Simple/simplePrintf/cuPrintf.cu>
#include<mpi.h>
#include "DHGraph.h"
```

```
#include<sys/time.h>
#include<unistd.h>
struct struct_hgraph {
int
*olddist ;//has to given size of property type
bool *updated ;//has to given size of property type
int
*dist ;//has to given size of property type
};
void alloc_extra_hgraph(DGGraph &hgraph, int flag,int npoints) {
struct struct_hgraph temp;
if(flag==0)cudaMalloc((void **)&(hgraph.extra),sizeof(struct struct_hgraph ));
cudaMemcpy(&temp,((struct struct_hgraph *)(hgraph.extra)),sizeof(struct
struct_hgraph ),cudaMemcpyDeviceToHost);
cudaMalloc( (void **)&(temp.olddist),sizeof(int )* hgraph.npoints);
cudaMalloc( (void **)&(temp.updated),sizeof(bool)* hgraph.npoints);
cudaMalloc( (void **)&(temp.dist),sizeof(int )* hgraph.npoints);
if(cudaMemcpy(hgraph.extra,&temp,sizeof(struct
struct_hgraph ),cudaMemcpyHostToDevice)!=cudaSuccess)printf("memcpyerror 0");
}
struct FALCbuffer{
int *vid;
int *dist;
};
int *tempdist;
int FALCrank;//rank of process
char partitionfile[100];//second input
char FALChostname[256];//name of host on which process running
int FALCsize;//total processes launched
MPI_Status *FALCstatus;//used for MPI_Recv
MPI_Request *FALCrequest;//Used for MPI_Isend
int *FALCsendsize;//send buffer size(for sending to remote machines)
int *FALCrecvsize;
int FALCmsgno;//message number for messages used in code
int FALCnamount;
struct FALCbuffer *FALCsendbuff,*FALCrecvbuff;//send and receive buffer for
synchronizing global state
//allocate buffer for communication
void FALCallocbuff(struct FALCbuffer *buff,int tot,int size){
struct FALCbuffer temp;
for(int i=0;i<tot;i++){
cudaMemcpy( &temp,&buff[i],sizeof(struct FALCbuffer),cudaMemcpyDeviceToHost);
cudaMalloc(&(temp.vid),sizeof(int)*size);
cudaMalloc(&(temp.dist),sizeof(int )*size);
cudaMemcpy(&buff[i],&temp,sizeof(struct FALCbuffer),cudaMemcpyHostToDevice);
}
cudaMalloc(&(tempdist),sizeof(int )*size);
}
__global__ void sendprefix(DGGraph hgraph,int *tempdist){
int id=threadIdx.x+blockDim.x*blockIdx.x;
if(id<(hgraph.localpoints+hgraph.remotepoints)){
tempdist[id]=((struct struct_hgraph *)(hgraph.extra))->dist[id];
}
}
__global__ void sendbuff(DGGraph hgraph,int *sendsize,struct FALCbuffer
*sendbuff,int *tempdist,int kk,int off,int totelems){
int id=threadIdx.x+blockDim.x*blockIdx.x;
if(id <totelems){
int loc=0;
int flag=0;
if( (( struct struct_hgraph *)(hgraph.extra))->dist[id+off]!
=tempdist[id+off])flag=1;
if(flag==1){
loc=atomicAdd(&sendsize[kk],1);
sendbuff[kk].vid[loc]=hgraph.remotevertexid[id+off];
sendbuff[kk].dist[loc]=(( struct struct_hgraph *)(hgraph.extra))->dist[id
+off];
tempdist[id+off]=(( struct struct_hgraph *)(hgraph.extra))->dist[id+off];
}
}
}
__global__ void update(DGGraph hgraph,struct FALCbuffer *recvbuff,int
FALCnamount,int kk){
int id=blockIdx.x*blockDim.x+threadIdx.x;
if(id <FALCnamount){
int vertex= recvbuff[kk].vid[id];
if( ( ( struct struct_hgraph * )(hgraph.extra))->dist[vertex] >
recvbuff[kk].dist[id])
( ( struct struct_hgraph * )(hgraph.extra))->dist[vertex] =
recvbuff[kk].dist[id];
}
}
```

```
--------------------------------------------------------------------------------
----------------
sssp.cu
--------------------------------------------------------------------------------
----------------
#include "sssp.h"
void FALCmpiinit(int argc,char **argv){
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &FALCrank);
MPI_Comm_size(MPI_COMM_WORLD, &FALCsize);
gethostname(FALChostname,255);
cudaMalloc(&FALCsendbuff,sizeof(struct FALCbuffer )*FALCsize);
cudaMalloc(&FALCrecvbuff,sizeof(struct FALCbuffer )*FALCsize);
cudaMalloc(&FALCsendsize,sizeof(int)*FALCsize);
cudaMalloc(&FALCrecvsize,sizeof(int)*FALCsize);
for(int i=0;i<FALCsize;i++){int temp=0;
cudaMemcpy(&FALCsendsize[i],&temp,sizeof(int),cudaMemcpyHostToDevice);}
FALCstatus=(MPI_Status *)malloc(sizeof(MPI_Status)*FALCsize);
FALCrequest=(MPI_Request *)malloc(sizeof(MPI_Request)*FALCsize);
}
__device__ int changed =0;
;
__global__ void relaxgraph ( DGGraph graph ,int FALCX)
{
int id= blockIdx.x * blockDim.x + threadIdx.x+FALCX;
if( id < graph.localpoints&& /*J74*/((struct struct_hgraph)(graph.extra))-
>updated[id]==true ){
((struct struct_hgraph *)(graph.extra))->updated[id]=false;

int falcft0=graph.index[id+1]-graph.index[id];
int falcft1=graph.index[id];
/*XX*/for(int falcft2=0;falcft2<falcft0;falcft2++){
int ut0=2*(falcft1+falcft2);
int ut1=graph.edges[ut0].ipe;
int ut2=graph.edges[ut0+1].ipe;
GMIN(&(((struct struct_hgraph *)(graph.extra))->dist/*xx*/[ut1]),((struct
struct_hgraph *)(graph.extra))->dist[id]+ut2,/**/changed);//rhs not null

}//foreach
}//end fun 0
}
__global__ void reset ( DGGraph graph ,int FALCX)
```

```
{
/* 0 xx*/int id= blockIdx.x * blockDim.x + threadIdx.x+FALCX;
if( id < graph.localpoints+graph.remotepoints){
((struct struct_hgraph *)(graph.extra))->dist[id]=1234567890;

((struct struct_hgraph *)(graph.extra))->olddist[id]=1234567890;

((struct struct_hgraph *)(graph.extra))->updated[id]=false;

}//end fun 0
}
__global__ void reset1 ( DGGraph graph ,int FALCX)
{
int id= blockIdx.x * blockDim.x + threadIdx.x+FALCX;
if( id < graph.localpoints){
if( ((struct struct_hgraph *)(graph.extra))->dist[id]<((struct struct_hgraph*)
(graph.extra))->olddist[id] )((struct struct_hgraph *)(graph.extra))-
>updated[id]=true;
((struct struct_hgraph *)(graph.extra))->olddist[id]=((struct struct_hgraph *)
(graph.extra))->dist[id];
}//end fun 0
}
/*X 0X*/void SSSP ( char * name ) /*XX*/
{
DGGraph hgraph ;

hgraph.readPointsN(partitionfile,FALCsize);
hgraph.makeNPartitionsMPI(name,FALCrank,FALCsize);
int hgraphflag=0;
alloc_extra_hgraph(hgraph,hgraphflag,hgraph.npoints);
FALCallocbuff(FALCsendbuff,FALCsize,hgraph.remotepoints);
FALCallocbuff(FALCrecvbuff,1,hgraph.npoints);
int TPB0=1024;
;
cudaSetDevice(FALCrank);
reset<<<hgraph.npoints/TPB0+1,TPB0>>>(hgraph,0);
cudaDeviceSynchronize();

if(FALCrank==0){
bool falcvt1;
falcvt1=true;


struct struct_hgraph temp0;
cudaMemcpy(&temp0,((struct struct_hgraph *)(hgraph.extra)),sizeof(struct
struct_hgraph ),cudaMemcpyDeviceToHost);
if(cudaMemcpy(&(temp0.updated[0]),&(falcvt1),sizeof(bool),cudaMemcpyHostToDevice
)!=cudaSuccess)printf("memcpyerror 6");
}
if(FALCrank==0){
int
falcvt2;
falcvt2=0;
struct struct_hgraph temp1;
cudaMemcpy(&temp1,((struct struct_hgraph *)(hgraph.extra)),sizeof(struct
struct_hgraph ),cudaMemcpyDeviceToHost);
if(cudaMemcpy(&(temp1.dist[0]),&(falcvt2),sizeof(int ),cudaMemcpyHostToDevice)!
=cudaSuccess)printf("memcpyerror 7");
}
while(1) {
int
falcvt3;
falcvt3=0;
if(cudaMemcpyToSymbol(changed,&(falcvt3),sizeof(int ),0,cudaMemcpyHostToDevice)!
=cudaSuccess)printf("memcpyerror 8");//val=1
sendprefix<<<(hgraph.localpoints+hgraph.remotepoints)/1024+1,1024>>>
(hgraph,tempdist);
cudaDeviceSynchronize();
relaxgraph<<<hgraph.localpoints/TPB0+1,TPB0>>>(hgraph,0);
cudaDeviceSynchronize();
for(int kk=1;kk<FALCsize;kk++){
int offstart,offend;
offstart=hgraph.offset[kk-1];
offend=hgraph.offset[kk];
sendbuff<<<(offend-offstart)/1024+1,1024>>>
(hgraph,FALCsendsize,FALCsendbuff,tempdist,kk-1,offstart,(offend-offstart));
}
cudaDeviceSynchronize();
for(int i=0;i<FALCsize;i++){
struct FALCbuffer temp;
if(i<FALCrank){
cudaMemcpy( &temp,&(FALCsendbuff[i]),sizeof(struct
FALCbuffer),cudaMemcpyDeviceToHost);
int temp1;
cudaMemcpy( &temp1,&(FALCsendsize[i]),sizeof(int),cudaMemcpyDeviceToHost);
MPI_Isend((temp.vid), temp1, MPI_INT, i ,FALCmsgno,
MPI_COMM_WORLD,&FALCrequest[i]);
}
if(i>FALCrank){
cudaMemcpy( &temp,&(FALCsendbuff[i-1]),sizeof(struct
FALCbuffer),cudaMemcpyDeviceToHost);
int temp1;
cudaMemcpy( &temp1,&(FALCsendsize[i-1]),sizeof(int),cudaMemcpyDeviceToHost);
MPI_Isend((temp.vid), temp1, MPI_INT, i
,FALCmsgno,MPI_COMM_WORLD,&FALCrequest[i-1]);
}
}
for(int i=0;i<FALCsize;i++){
struct FALCbuffer temp;
if(i<FALCrank){
cudaMemcpy( &temp,&(FALCsendbuff[i]),sizeof(struct
FALCbuffer),cudaMemcpyDeviceToHost);
int temp1;
cudaMemcpy( &temp1,&(FALCsendsize[i]),sizeof(int),cudaMemcpyDeviceToHost);
MPI_Isend((temp.dist), temp1, MPI_INT, i ,FALCmsgno,
MPI_COMM_WORLD,&FALCrequest[i]);
}
if(i>FALCrank){
cudaMemcpy( &temp,&(FALCsendbuff[i-1]),sizeof(struct
FALCbuffer),cudaMemcpyDeviceToHost);
int temp1;
cudaMemcpy( &temp1,&FALCsendsize[i-1],sizeof(int),cudaMemcpyDeviceToHost);
MPI_Isend((temp.dist), temp1, MPI_INT, i ,FALCmsgno,
MPI_COMM_WORLD,&FALCrequest[i-1]);
}
}
for(int i=0;i<FALCsize;i++){
struct FALCbuffer temp;
if(i<FALCrank){
cudaMemcpy( &temp,&(FALCrecvbuff[0]),sizeof(struct
FALCbuffer),cudaMemcpyDeviceToHost);
MPI_Recv(temp.vid,hgraph.npoints, MPI_INT,i,
FALCmsgno,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
MPI_Recv(temp.dist,hgraph.npoints, MPI_INT,i, FALCmsgno,
MPI_COMM_WORLD,&FALCstatus[i]);
MPI_Get_count(&FALCstatus[kk-1], MPI_INT, &FALCnamount);
}
if(i>FALCrank){
```

```
cudaMemcpy( &temp,&(FALCrecvbuff[0]),sizeof(struct
FALCbuffer),cudaMemcpyDeviceToHost);
MPI_Recv(temp.vid,hgraph.npoints, MPI_INT,i,
FALCmsgno,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
MPI_Recv(temp.dist,hgrap.npoints, MPI_INT,i, FALCmsgno,
MPI_COMM_WORLD,&FALCstatus[i-1]);
MPI_Get_count(&FALCstatus[kk-1], MPI_INT, &FALCnamount);
}
update<<< FALCnamount/1024+1,1024>>>(hgraph,FALCrecvbuff,FALCnamount,0);
}
}//changed should be synchronized as it is a global var
for(int i=0;i<FALCsize;i++){
int temp=0;
cudaMemcpy(&FALCsendsize[i],&temp,sizeof(int),cudaMemcpyHostToDevice);
}
cudaDeviceSynchronize();
int tempchanged=0,tempchanged1=0;
cudaMemcpyFromSymbol(&tempchanged,changed,sizeof(int),0,cudaMemcpyDeviceToHost);
for(int i=0;i<NPARTS;i++){
if(i!=rank)MPI_Isend(&tempchanged, 1, MPI_INT, i,messageno,
MPI_COMM_WORLD,&request[i]);
}
for(int i=0;i<NPARTS;i++){
if(i!=rank)MPI_Recv(&tempchanged1,1, MPI_INT, i,messageno,
MPI_COMM_WORLD,MPI_STATUS_IGNORE);
tempchanged=tempchanged+changed1;
}
if(tempchanged==0)break;

reset1<<<hgraph.localpoints/TPB0+1,TPB0>>>(hgraph,0);
cudaDeviceSynchronize();
}//end
struct struct_hgraph temp2;
cudaMemcpy((void *)&temp2,hgraph.extra,sizeof( struct
struct_hgraph ),cudaMemcpyDeviceToHost);

struct struct_hgraph temp3;
cudaMemcpy((void *)&temp3,hgraph.extra,sizeof( struct
struct_hgraph ),cudaMemcpyDeviceToHost);
int *temp5=(int *) malloc(sizeof(int )*hgraph.npoints);
struct struct_hgraph temp6;
cudaMemcpy((void *)&temp6,hgraph.extra,sizeof(struct
struct_hgraph ),cudaMemcpyDeviceToHost);
cudaMemcpy(temp5, temp6.dist,sizeof(int )
*hgraph.npoints,cudaMemcpyDeviceToHost);
int max=0;
for (int i =0;i<hgraph.localpoints;i++) {
if(max <temp5[i])max=temp5[i];
}//endfor
printf("%d\n",max);
return ;
}//end fun 0
int main ( int argc ,char * argv [ ] ) /*XX*/
{
FALCsize=atoi(argv[3]);
FALCmpiinit(argc,argv);
sprintf(partitionfile,"%s",argv[2]);
SSSP(argv[1]);//rhs not null

MPI_Finalize();
}
```