CS 406 Project Presentation

Ahmet Enes Deveci, Alper Giray, Efe Şencan, Gökberk Yar, Zeynep Özge Ergin

Content

- Multi-Core CPU Implementation (OpenMP)
- GPU Implementation (CUDA)
  - Single GPU Implementation
  - Multi-GPU Implementation
  - Multi-GPU & CPU Implementation
  - Multi-GPU & CPU Dynamic Work Queue Implementation

Multi-core CPU Implementation with OpenMP

- Non Recursive DFS with limited depth, for each vertex.
- Outer for loop is parallelized among CPU threads.

```
double start, end;
start = omp_get_wtime();
int *arr = new int[nov];

#pragma omp parallel num_threads(numThreads)
{
    if(n==3){
        #pragma omp for schedule(guided)
        for(int i = 0; i < nov; i++){
            arr[i] =sparse3(xadj, adj, i);
        }
    }
    else if(n==4){
        #pragma omp for schedule(guided)
        for(int i = 0; i < nov; i++){
            arr[i] = sparse4(xadj, adj, i);
        }
    }else if(n==5){
        #pragma omp for schedule(guided)
        for(int i = 0; i < nov; i++){
            arr[i] = sparse5(xadj, adj, i);
        }
    }
}
end = omp_get_wtime();
```

```
int sparse3(int xadj[], int adj[], int index) //vert: b
{
    int localcount = 0;
    int s0 = xadj[index];
    int e0 = xadj[index+1];
    for(int i=s0; i < e0; i++){
    // 0 --> 1
        int neighbour_1 = adj[i];
        int s1  = xadj[neighbour_1];
        int e1  = xadj[neighbour_1+1];

        for(int j=s1;j < e1; j++){
            // 0 --> 1 --> 2
            int neighbour_2 =  adj[j];
            if (neighbour_2 == index) continue;
            int s2  = xadj[neighbour_2];
            int e2  = xadj[neighbour_2+1];

            for(int k=s2; k < e2; k++){
                // 0 --> 1 --> 2 --> 3
                int neighbour_3 =  adj[k];
                if (neighbour_3 == index){
                    localcount+=1;
                    break;
                }
            }
        }
    }
    return localcount;
}
```

Multi-core CPU with OpenMP Results

- Highly Scalable
- Sparse Matrix performs better (look at the speedups amazon vs dblp)

OPENMP				
K=3	Thread Count	Duration	Speedup	Efficiency
amazon	1	0.603501	1	1
	2	0.310298	1.944907798	0.9724538992
	4	0.156428	3.858011353	0.9645028384
	8	0.0786051	7.677631604	0.9597039505
	16	0.0402396	14.99768884	0.9373555527
dblp	1	1.8533	1	1
	2	1.12785	1.643214967	0.8216074833
	4	0.659231	2.8113059	0.7028264751
	8	0.353933	5.236301786	0.6545377232
	16	0.178739	10.36874997	0.6480468728
K=4	Thread Count	Duration	Speedup	Efficiency
amazon	1	4.55978	1	1
	2	2.29525	1.986615837	0.9933079185
	4	1.19381	3.819519019	0.9548797547
	8	0.578946	7.876002252	0.9845002815
	16	0.287246	15.8741288	0.9921330497
dblp	1	66.5606	1	1
	2	34.1796	1.947377968	0.9736889841
	4	20.5717	3.235542031	0.8088855078
	8	12.5448	5.305831898	0.6632289873
	16	9.59927	6.9339231	0.4333701938
K=5	Thread Count	Duration	Speedup	Efficiency
amazon	1	43.1156	1	1
	2	21.9676	1.962690508	0.9813452539
	4	10.9534	3.936275494	0.9840688736
	8	5.42065	7.953953862	0.9942442327
	16	2.86459	15.05122897	0.9407018107
dblp	1	-	-	-
	2	-	-	-
	4	-	-	-
	8	-	-	-
	32	977	-	-

Table 1: Multicore OpenMP Implementation

Single GPU Implementation

```
if (n==3)       kernel3<<<numBlock, THREADS_PER_BLOCK>>>(adj_d, xadj_d, output_d, nov);
else if (n==4)  kernel4<<<numBlock, THREADS_PER_BLOCK>>>(adj_d, xadj_d, output_d, nov);
else if (n==5)  kernel5<<<numBlock, THREADS_PER_BLOCK>>>(adj_d, xadj_d, output_d, nov);
```

```
int *output_h = new int[nov];
int numBlock = (nov + THREADS_PER_BLOCK - 1) / THREADS_PER_BLOCK;
```

- Total Thread Count : Vertex Count
- Each Thread is responsible for one vertex.
- Similar algorithm to CPU.

```
__global__ void kernel3(int* adj, int* xadj, int* output, int nov){
  int index = threadIdx.x + (blockIdx.x * blockDim.x);
  if(index < nov){
      //int *marked = new int[n];
      //memset(marked, -1, n * sizeof(int)); // bu belki silinebilir
      int localcount = 0;
      // int round = 0;

      // 0-->
      int s0 = xadj[index];
      int e0 = xadj[index+1];

      for(int i=s0; i < e0; i++){
        // 0 --> 1

        int neighbour_1 = adj[i];
        int s1  = xadj[neighbour_1];
        int e1  = xadj[neighbour_1+1];

        for(int j=s1;j < e1; j++){
          // 0 --> 1 --> 2

          int neighbour_2 =  adj[j];
          if (neighbour_2 == index) continue;
          int s2  = xadj[neighbour_2];
          int e2  = xadj[neighbour_2+1];

          for(int k=s2; k < e2; k++){

            // 0 --> 1 --> 2 --> 3
            int neighbour_3 =  adj[k];
            if (neighbour_3 == index){
              localcount+=1;
              break;
            }
          }
        }
      }
      output[index] = localcount;
    }
}
```

Single GPU Results

- Problem : Vertices with many neighbours.
- Work unbalance within warps. (idling)
- As search space increase, the work unbalance increases

Non Recursive		
K=3	**Duration**	**Speedup**
amazon	0.045381	13.29853904
dblp	0.487654	3.800440476

K=4	**Duration**	**Speedup**
amazon	0.514845	8.856607328
dblp	44.543335	1.494288652

K=5	**Duration**	**Speedup**
amazon	6.447344	6.687342881
dblp	-	-

Table 2: GPU Non-Recursive

Multi-GPU Implementation

- Machine Nebula
- 4 GPUs: 2 Faster and 2 Slower GPUs
- 4 OMP threads each control one GPU.
- Data splitted statically and evenly among all GPUs.

```
#pragma omp parallel num_threads(MY_GPU_COUNT)
{

  int threadId=omp_get_thread_num ();
  // cout<< threadId<<endl;

  int novForThread = (nov+MY_GPU_COUNT-1)/MY_GPU_COUNT;


  int novStart =   novForThread * threadId;
  int novEnd   =  novForThread * (threadId+1);
  if (novEnd > nov) novEnd = nov;
  int numBlock = (novEnd-novStart + THREADS_PER_BLOCK-1) / THREADS_PER_BLOCK;
  // printf("nov s %d e %d \n", novStart,novEnd);


  cudaSetDevice(threadId);

  int *adj_d;
  int *xadj_d;
  int *output_d;
  cudaEvent_t start, stop;
  float elapsedTime;

  gpuErrchk(cudaMalloc((void**)&adj_d, (nnz) * sizeof(int)));
  gpuErrchk(cudaMalloc((void**)&xadj_d, (nov + 1) * sizeof(int)));

  gpuErrchk(cudaMalloc((void**)&output_d, (novEnd-novStart) * sizeof(int)));

  //gpuErrchk(cudaMallocHost((void **)&output_h, (nov) * sizeof(int)));

  gpuErrchk(cudaMemcpy(adj_d, adj, (nnz) * sizeof(int), cudaMemcpyHostToDevice));
  gpuErrchk(cudaMemcpy(xadj_d, xadj, (nov + 1) * sizeof(int), cudaMemcpyHostToDevice));

  cudaEventCreate(&start);
  cudaEventRecord(start, 0);
  double start_gpu = omp_get_wtime();
  cudaStream_t stream1;
  cudaStreamCreate ( &stream1) ;

    // printf("threadId entry to kernel %d GPU \n", threadId );
    if      (n==3)kernel3<<<numBlock, THREADS_PER_BLOCK,0,stream1>>>(adj_d, xadj_d, output_d, novEnd,novStart);
    else if (n==4)kernel4<<<numBlock, THREADS_PER_BLOCK,0,stream1>>>(adj_d, xadj_d, output_d, novEnd,novStart);
    else if (n==5)kernel5<<<numBlock, THREADS_PER_BLOCK,0,stream1>>>(adj_d, xadj_d, output_d, novEnd,novStart);
```

Multi-GPU Results

- The slowest result is taken as final result.
- Load Balance Problem
- Idle
- Uneven GPUs

Multigpu - Non Recursive		
K=3	Duration	Speedup
amazon	0.030755	19.62285807
dblp	0.49196	3.767176193

K=4	Duration	Speedup
amazon	0.372918	12.2272993
dblp	57.406826	1.159454452

K=5	Duration	Speedup
amazon	5.285642	8.157116959
dblp	-	-

Table 3: Multi-GPU Non-Recursive

Multi-GPU+CPU Implementation

```
if(threadId <=3)
{
  int novStart = GPU_MULTIPLIER * novForThread * threadId;
  int novEnd   = GPU_MULTIPLIER * novForThread * (threadId+1);
  if (novEnd > nov) novEnd = nov;
  int numBlock = (novEnd-novStart + THREADS_PER_BLOCK-1) / THREADS_PER_BLOCK;
  // printf("nov s %d e %d \n", novStart,novEnd);


  cudaSetDevice(threadId);

  int *adj_d;
  int *xadj_d;
  int *output_d;
  cudaEvent_t start, stop;
  float elapsedTime;

  gpuErrchk(cudaMalloc((void**)&adj_d, (nnz) * sizeof(int)));
  gpuErrchk(cudaMalloc((void**)&xadj_d, (nov + 1) * sizeof(int)));

  gpuErrchk(cudaMalloc((void**)&output_d, (novEnd-novStart) * sizeof(int)));

  //gpuErrchk(cudaMallocHost((void **)&output_h, (nov) * sizeof(int)));

  gpuErrchk(cudaMemcpy(adj_d, adj, (nnz) * sizeof(int), cudaMemcpyHostToDevice));
  gpuErrchk(cudaMemcpy(xadj_d, xadj, (nov + 1) * sizeof(int), cudaMemcpyHostToDevice));

  cudaEventCreate(&start);
  cudaEventRecord(start, 0);
  double start_gpu = omp_get_wtime();
  cudaStream_t stream1;
  cudaStreamCreate ( &stream1) ;

    // printf("threadId entry to kernel %d GPU \n", threadId );
  if      (n==3)kernel3<<<numBlock, THREADS_PER_BLOCK,0,stream1>>>(adj_d, xadj_d, output_d, novEnd,novStart);
  else if (n==4)kernel4<<<numBlock, THREADS_PER_BLOCK,0,stream1>>>(adj_d, xadj_d, output_d, novEnd,novStart);
   else if (n==5)kernel5<<<numBlock, THREADS_PER_BLOCK,0,stream1>>>(adj_d, xadj_d, output_d, novEnd,novStart);
  //combination<<<numBlocks, threadsPerBlock>>>(adj_d, xadj_d, output_d, n, nov);
  // printf("threadId exit to kernel %d GPU \n", threadId );
  double end_gpu = omp_get_wtime();
```

GPU PART

```
else{
  // printf("Entered \n" );

  // int novForThread = (nov+PARALEL_THREAD_COUNT-1)/PARALEL_THREAD_COUNT;
  // int novStart = novForThread * threadId;
  // int novEnd   = novForThread * (threadId+1);
  // if (novEnd> nov) novEnd = nov;
  // int numBlock = (novEnd-novStart + THREADS_PER_BLOCK-1) / THREADS_PER_BLOCK;
  int novStart = 4 * GPU_MULTIPLIER*novForThread +  1 * novForThread * (threadId-4);
  int novEnd   =  novStart + 1*  novForThread ;
  if (novEnd> nov) novEnd = nov;


  // printf("nov s %d e %d  -cpu \n", novStart,novEnd);

  bool *marked = new bool[nov];
  memset(marked, false, nov * sizeof(bool)); // bu belki silinebilir

  double start_thread = omp_get_wtime();
  for(int i = novStart; i < novEnd; i++){
      int localcount = 0;
      DFS_sparse(xadj, adj, marked, n - 1, i, i, localcount);
      output_h[i ] = localcount;


  }
  double end_thread = omp_get_wtime();
  if(flag == 1)     printf("Took %f secs \n", end_thread -start_thread );
```

CPU PART

```
#define THREADS_PER_BLOCK 256
#define PARALEL_THREAD_COUNT 32
#define GPU_MULTIPLIER 10
#define PARALEL_CPU 28
```

Multi-GPU+CPU Results

- Total 32 threads : 4 controls GPU, 28 CPU threads.
- GPU Multiplier (GPUs take 10 times more work than CPU)
- Work Distributed Statically
- Load Balancing Problem
- Idle

Multigpu + CPU Non Recursive		
K=3	Duration	Speedup
amazon	0.037041	16.29278367
dblp	0.239933	7.724239684

K=4	Duration	Speedup
amazon	0.274205	16.62909137
dblp	22.658237	2.93758954

K=5	Duration	Speedup
amazon	4.201431	10.2621226
dblp	2200	-

Table 4: Multi-GPU + CPU Non-Recursive

Multi-GPU+CPU Dynamic Work Queue Implementation

```
using namespace std;
#define THREADS_PER_BLOCK 512
#define PARALEL_THREAD_COUNT 32
#define GPU_MULTIPLIER 8
#define PARALEL_CPU 28
#define CHUNK_SIZE_OUR 64
#define OUR_GPU_COUNT 4
```

- Gpu multiplier
- Chunk Size
- Dynamic work distribution

```
while(true){
  int thisChunk;

  #pragma omp critical
  {
    thisChunk=currentChunk; //thisChunk is different on everyone.
    currentChunk+= GPU_MULTIPLIER;

  }
  if (thisChunk  >= totalChunck) break;

  int novStart   = thisChunk * CHUNK_SIZE_OUR;

  int novEnd =  novStart + GPU_MULTIPLIER * CHUNK_SIZE_OUR;

  // printf("nov s %d e %d \n", novStart,novEnd);
  if(novEnd>nov) novEnd=nov;

  int numBlock = (novEnd-novStart + THREADS_PER_BLOCK-1) / THREADS_PER_BLOCK;
```

Multi-GPU+CPU Dynamic Work Queue Implementation

- 32 thread : 4 GPU + 28 CPU
- Communication Cost is increased when chunk size small
- Idling is increased when chunk size is large

Multigpu + CPU Non Recursive Dynamic Workload		
K=3	**Duration**	**Speedup**
amazon	2.076866	0.2905825412
dblp	3.700613	0.50080892

K=4	**Duration**	**Speedup**
amazon	5.658017	0.8058971898
dblp	10.617888	6.268723121

K=5	**Duration**	**Speedup**
amazon	8.319901	5.182225125
dblp	-	-

Table 5: Multi-GPU + CPU Non-Recursive Dynamic Workload

OPENMP				
K=3	Thread Count	Duration	Speedup	Efficiency
amazon	1	0.603501	1	1
	2	0.310298	1.944907798	0.9724538992
	4	0.156428	3.858011353	0.9645028384
	8	0.0786051	7.677631604	0.9597039505
	16	0.0402396	14.99768884	0.9373555527
dblp	1	1.8533	1	1
	2	1.12785	1.643214967	0.8216074833
	4	0.659231	2.8113059	0.7028264751
	8	0.353933	5.236301786	0.6545377232
	16	0.178739	10.36874997	0.6480468728
K=4	Thread Count	Duration	Speedup	Efficiency
amazon	1	4.55978	1	1
	2	2.29525	1.986615837	0.9933079185
	4	1.19381	3.819519019	0.9548797547
	8	0.578946	7.876002252	0.9845002815
	16	0.287246	15.8741288	0.9921330497
dblp	1	66.5606	1	1
	2	34.1796	1.947377968	0.9736889841
	4	20.5717	3.235542031	0.8088855078
	8	12.5448	5.305831898	0.6632289873
	16	9.59927	6.9339231	0.4333701938
K=5	Thread Count	Duration	Speedup	Efficiency
amazon	1	43.1156	1	1
	2	21.9676	1.962690508	0.9813452539
	4	10.9534	3.936275494	0.9840688736
	8	5.42065	7.953953862	0.9942442327
	16	2.86459	15.05122897	0.9407018107
dblp	1	-	-	-
	2	-	-	-
	4	-	-	-
	8	-	-	-
	32	977	-	-

Table 1: Multicore OpenMP Implementation

Non Recursive		
K=3	Duration	Speedup
amazon	0.045381	13.29853904
dblp	0.487654	3.800440476
K=4	Duration	Speedup
amazon	0.514845	8.856607328
dblp	44.543335	1.494288652
K=5	Duration	Speedup
amazon	6.447344	6.687342881
dblp	-	-

Table 2: GPU Non-Recursive

Multigpu - Non Recursive		
K=3	Duration	Speedup
amazon	0.030755	19.62285807
dblp	0.49196	3.767176193
K=4	Duration	Speedup
amazon	0.372918	12.2272993
dblp	57.406826	1.159454452
K=5	Duration	Speedup
amazon	5.285642	8.157116959
dblp	-	-

Table 3: Multi-GPU Non-Recursive

Multigpu + CPU Non Recursive		
K=3	Duration	Speedup
amazon	0.037041	16.29278367
dblp	0.239933	7.724239684
K=4	Duration	Speedup
amazon	0.274205	16.62909137
dblp	22.658237	2.93758954
K=5	Duration	Speedup
amazon	4.201431	10.2621226
dblp	2200	-

ble 4: Multi-GPU + CPU Non-Recursive

Multigpu + CPU Non Recursive Dynamic Workload		
K=3	Duration	Speedup
amazon	2.076866	0.2905825412
dblp	3.700613	0.50080892
K=4	Duration	Speedup
amazon	5.658017	0.8058971898
dblp	10.617888	6.268723121
K=5	Duration	Speedup
amazon	8.319901	5.182225125
dblp	-	-

Table 5: Multi-GPU + CPU Non-Recursive Dynamic Workload

Best among GPU

32 threads : 4 GPU + 28 CPU

Baseline 1 thread CPU

Reproducing Results

Make file command
for each case

```
run_gpu_nonrec_amz_3: GPU_Non_Recursive_Implementation
	./non_recursive_out amazon.txt 3 0

run_gpu_nonrec_amz_4: GPU_Non_Recursive_Implementation
	./non_recursive_out amazon.txt 4 0

run_gpu_nonrec_amz_5: GPU_Non_Recursive_Implementation
	./non_recursive_out amazon.txt 5 0

run_gpu_nonrec_dblp_3: GPU_Non_Recursive_Implementation
	./non_recursive_out dblp.txt 3 0

run_gpu_nonrec_dblp_4: GPU_Non_Recursive_Implementation
	./non_recursive_out dblp.txt 4 0

run_gpu_nonrec_dblp_5: GPU_Non_Recursive_Implementation
	./non_recursive_out dblp.txt 5 0

run_multigpu_non_rec_amz_3: Multi_GPU_Non_Recursive_Implementation
	./multi_gpu_out  amazon.txt 3 0

run_multigpu_non_rec_amz_4: Multi_GPU_Non_Recursive_Implementation
	./multi_gpu_out  amazon.txt 4 0
```