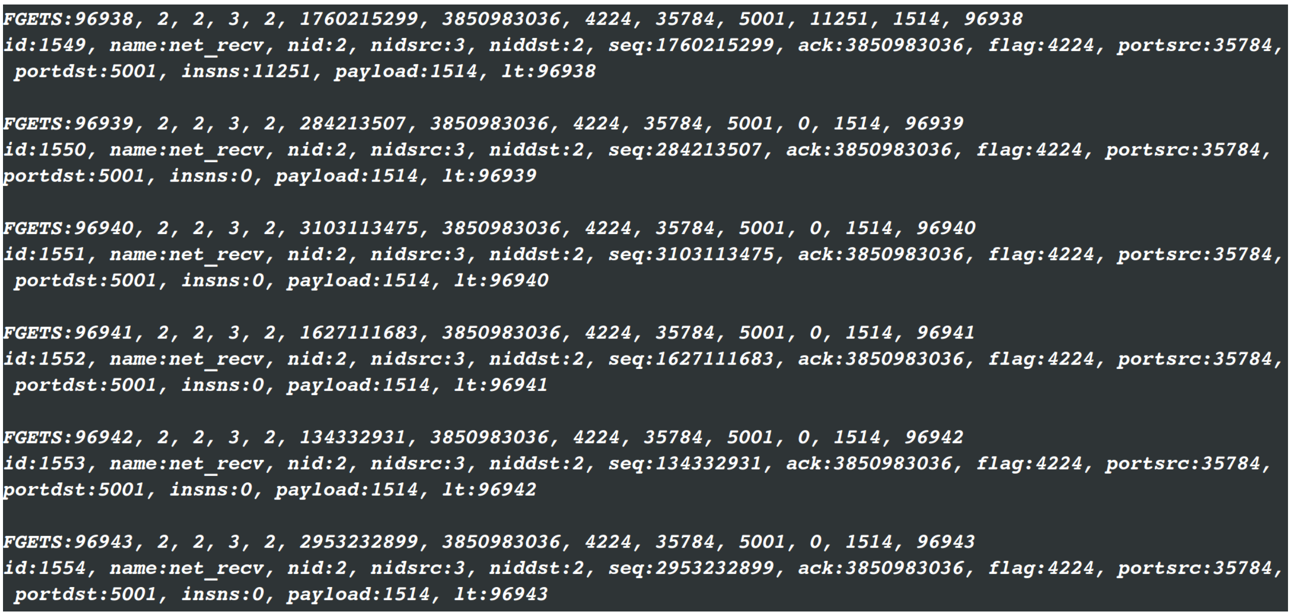
b02902080 郭傳駿
b02902100 康耀文

parallel & distributed computing finial report

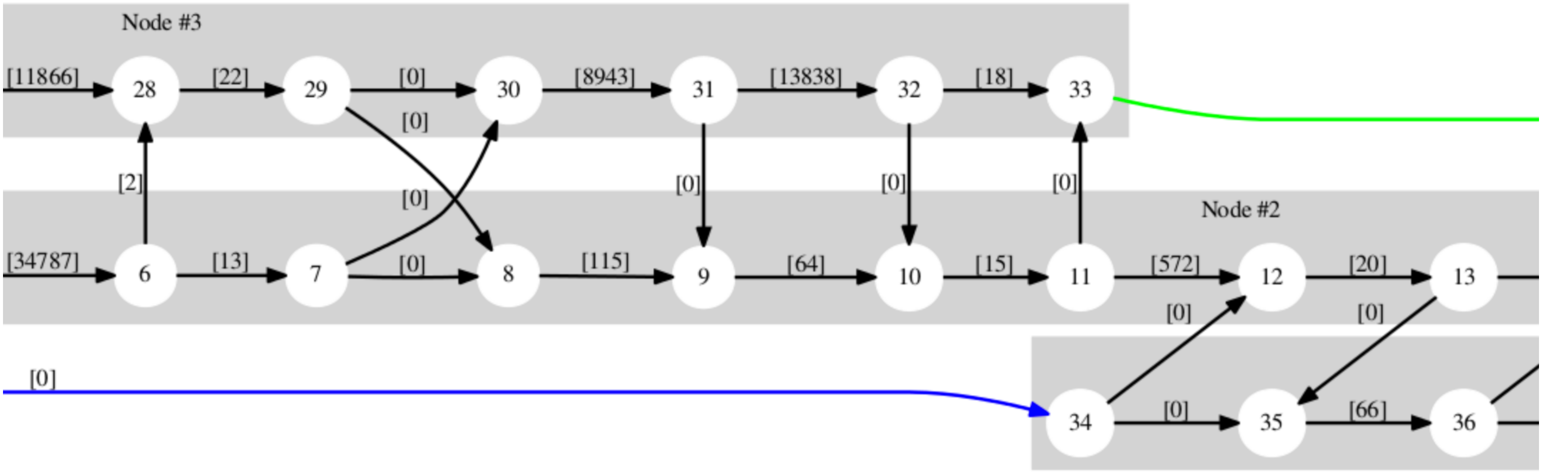
Mancos

- MICRO ARCHITECTURE AND NETWORK CO-SIMULATOR
- TRACE PACKET FLOW ACTIVITY
- APPLICATION: KEEPING TRACK OF
  - NETWORK FLOW
  - TRAFFIC FLOW
  - CASH FLOW
- INPUT: LOG FILE WITH ATTRIBUTES



```
FGETS:96938, 2, 2, 3, 2, 1760215299, 3850983036, 4224, 35784, 5001, 11251, 1514, 96938
id:1549, name:net_recv, nid:2, nidsrc:3, niddst:2, seq:1760215299, ack:3850983036, flag:4224, portsrc:35784,
 portdst:5001, insns:11251, payload:1514, lt:96938

FGETS:96939, 2, 2, 3, 2, 284213507, 3850983036, 4224, 35784, 5001, 0, 1514, 96939
id:1550, name:net_recv, nid:2, nidsrc:3, niddst:2, seq:284213507, ack:3850983036, flag:4224, portsrc:35784,
portdst:5001, insns:0, payload:1514, lt:96939

FGETS:96940, 2, 2, 3, 2, 3103113475, 3850983036, 4224, 35784, 5001, 0, 1514, 96940
id:1551, name:net_recv, nid:2, nidsrc:3, niddst:2, seq:3103113475, ack:3850983036, flag:4224, portsrc:35784,
 portdst:5001, insns:0, payload:1514, lt:96940

FGETS:96941, 2, 2, 3, 2, 1627111683, 3850983036, 4224, 35784, 5001, 0, 1514, 96941
id:1552, name:net_recv, nid:2, nidsrc:3, niddst:2, seq:1627111683, ack:3850983036, flag:4224, portsrc:35784,
 portdst:5001, insns:0, payload:1514, lt:96941

FGETS:96942, 2, 2, 3, 2, 134332931, 3850983036, 4224, 35784, 5001, 0, 1514, 96942
id:1553, name:net_recv, nid:2, nidsrc:3, niddst:2, seq:134332931, ack:3850983036, flag:4224, portsrc:35784,
portdst:5001, insns:0, payload:1514, lt:96942

FGETS:96943, 2, 2, 3, 2, 2953232899, 3850983036, 4224, 35784, 5001, 0, 1514, 96943
id:1554, name:net_recv, nid:2, nidsrc:3, niddst:2, seq:2953232899, ack:3850983036, flag:4224, portsrc:35784,
 portdst:5001, insns:0, payload:1514, lt:96943
```

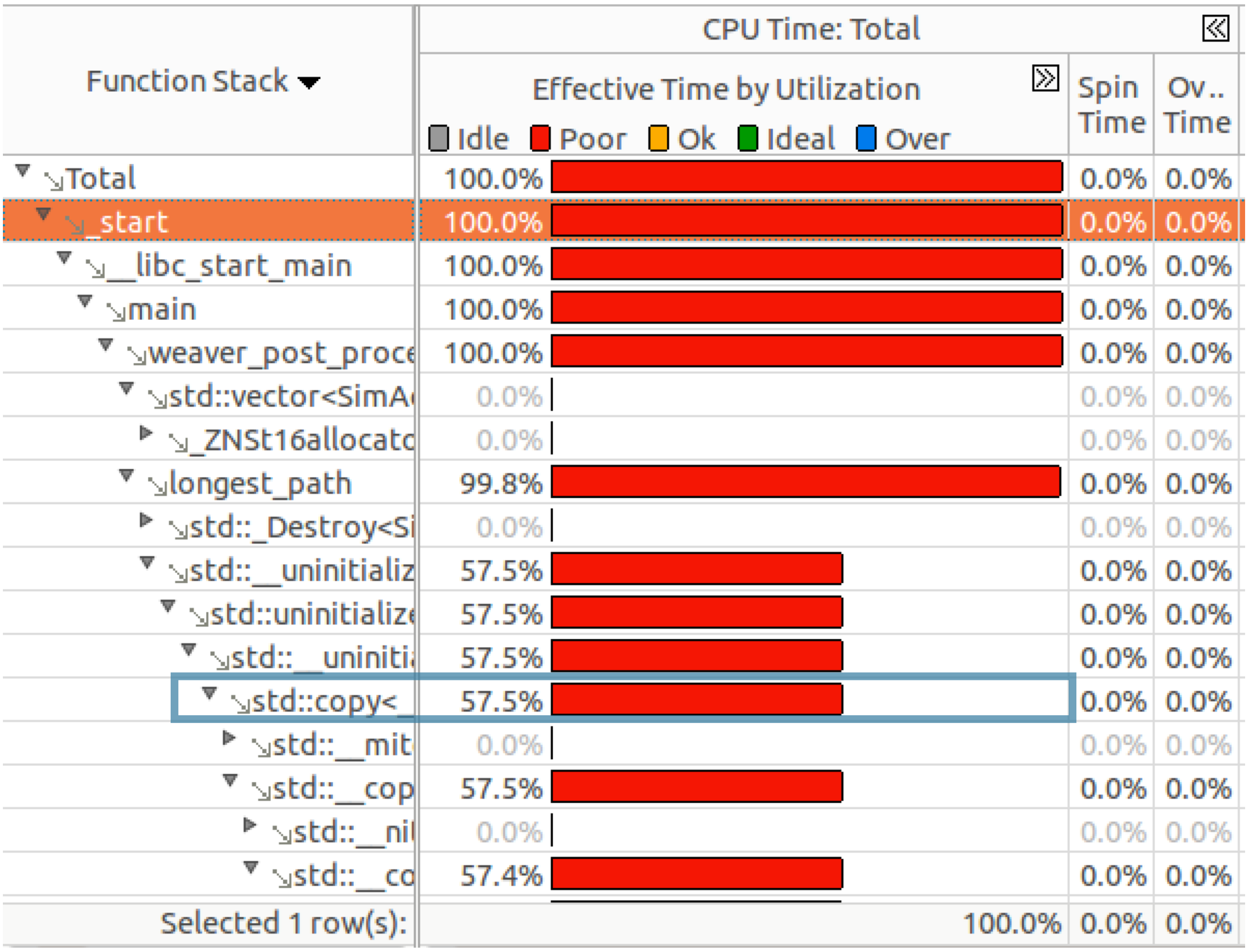
- OUTPUT: FLOW GRAPH



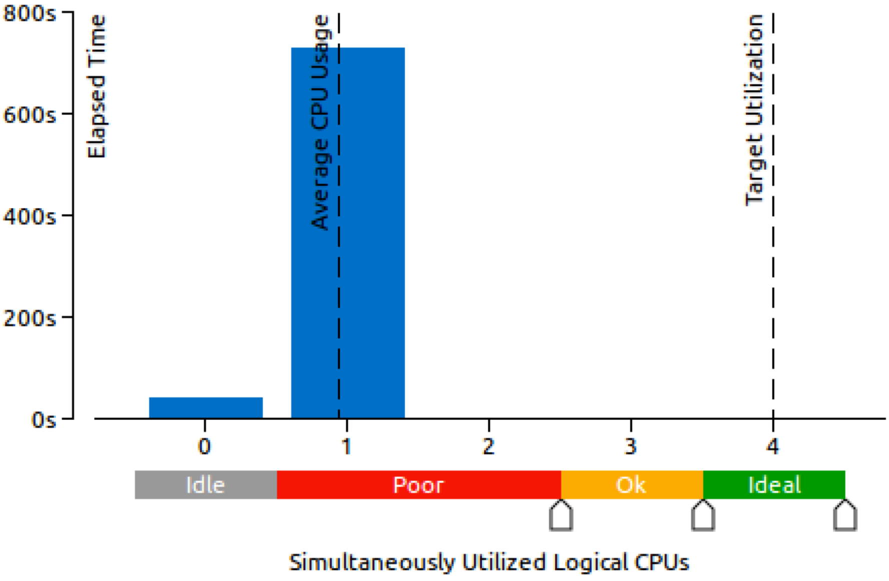
Why do we parallelise it?

The application must be scalable in reasonable time in order to deal with real world network activity between thousands of end-devices(nodes).

Vtune profile of the original version

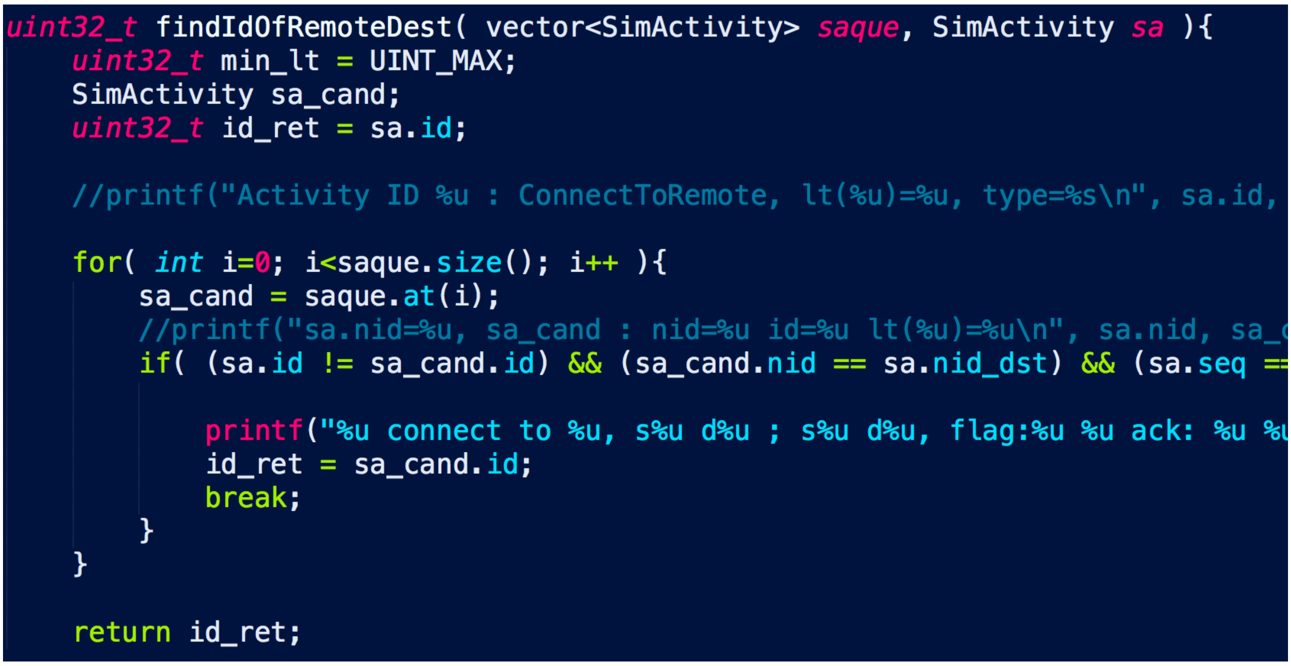


Function Stack ▾	CPU Time: Total		
	Effective Time by Utilization ■ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Spin Time	Ov.. Time
▾ ↘Total	100.0%	0.0%	0.0%
▾ ↘_start	100.0%	0.0%	0.0%
▾ ↘__libc_start_main	100.0%	0.0%	0.0%
▾ ↘main	100.0%	0.0%	0.0%
▾ ↘weaver_post_proce	100.0%	0.0%	0.0%
▾ ↘std::vector<SimA	0.0%	0.0%	0.0%
▸ ↘_ZNSt16allocato	0.0%	0.0%	0.0%
▾ ↘longest_path	99.8%	0.0%	0.0%
▸ ↘std::_Destroy<Si	0.0%	0.0%	0.0%
▾ ↘std::__uninitializ	57.5%	0.0%	0.0%
▾ ↘std::uninitialize	57.5%	0.0%	0.0%
▾ ↘std::__uniniti	57.5%	0.0%	0.0%
▾ ↘std::copy<	57.5%	0.0%	0.0%
▸ ↘std::__mit	0.0%	0.0%	0.0%
▾ ↘std::__cop	57.5%	0.0%	0.0%
▸ ↘std::__nil	0.0%	0.0%	0.0%
▾ ↘std::__co	57.4%	0.0%	0.0%
Selected 1 row(s):	100.0%	0.0%	0.0%



main bottleneck:

- function find_id_of_remote_dest
- find activity pairs(send/receive) over all activities in all nodes
- requires O(N*N) Time N= total num of activity



```
uint32_t findIdOfRemoteDest( vector<SimActivity> saque, SimActivity sa ){
    uint32_t min_lt = UINT_MAX;
    SimActivity sa_cand;
    uint32_t id_ret = sa.id;

    //printf("Activity ID %u : ConnectToRemote, lt(%u)=%u, type=%s\n", sa.id,

    for( int i=0; i<saque.size(); i++ ){
        sa_cand = saque.at(i);
        //printf("sa.nid=%u, sa_cand : nid=%u id=%u lt(%u)=%u\n", sa.nid, sa_c
        if( (sa.id != sa_cand.id) && (sa_cand.nid == sa.nid_dst) && (sa.seq ==

            printf("%u connect to %u, s%u d%u ; s%u d%u, flag:%u %u ack: %u %u
            id_ret = sa_cand.id;
            break;
        }
    }

    return id_ret;
```

code optimization:

1. KEEP TRACK OF EXISTING NODE AND FIRST ID OF EACH NODE

- int first_id[10] = {1};
- int node_id[10] = {0};
- if(flag == 0){

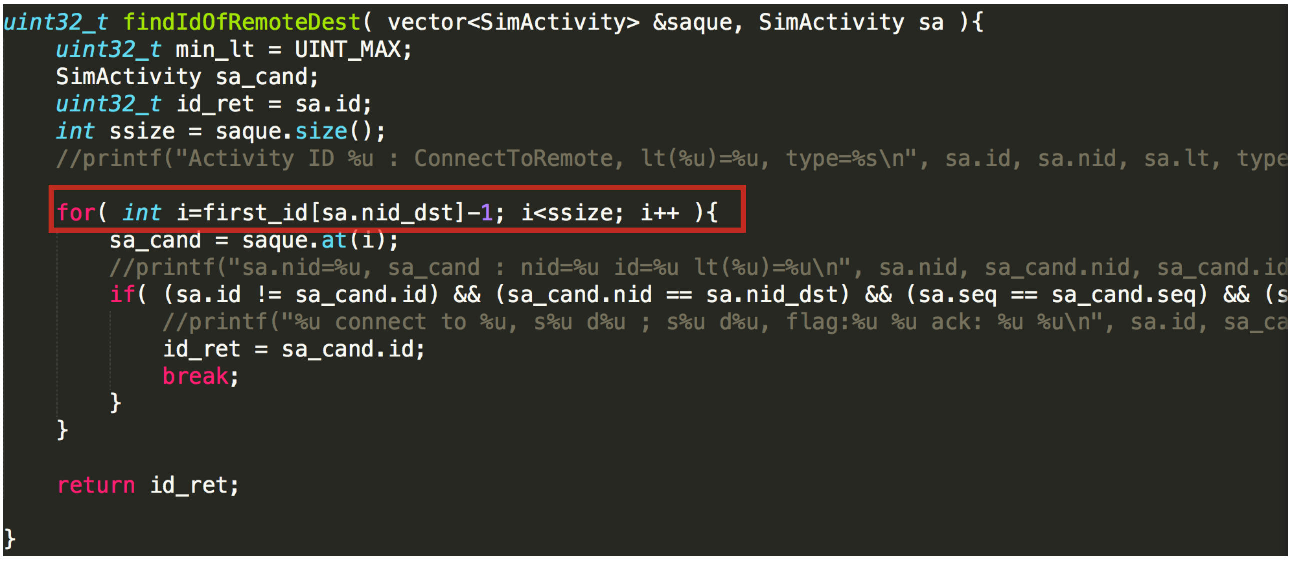
```
                                        first_id[n] = global_id;
                               flag = 1;
                             }
```

2. ACCORDING TO VTUNE ANALYSIS, THE COPYING OF “SAQUE” TAKES A LOT OF TIME

- we changed the function parameter from call by value to call by reference

3. IN FUNCTION FIND_ID_OF_REMOTE_DEST

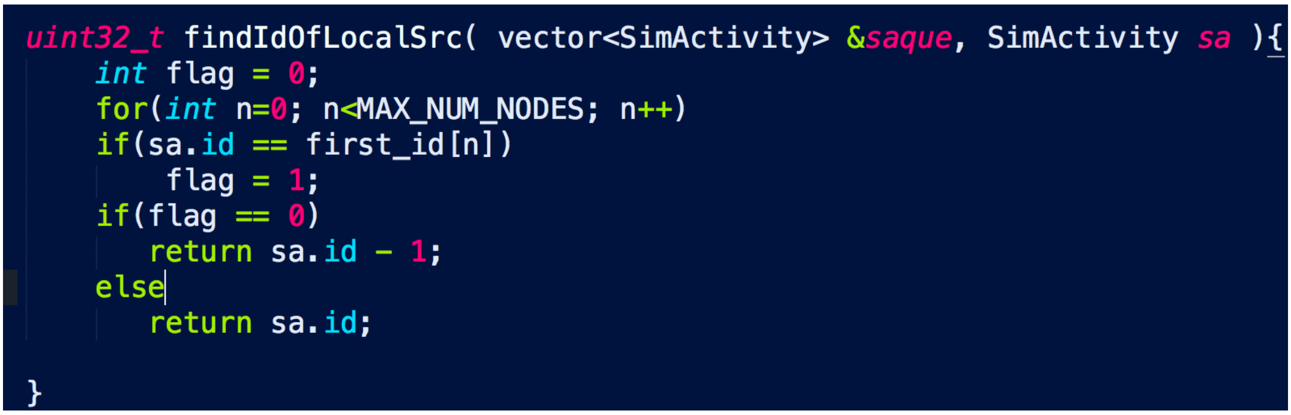
- search for pairs only in the section of the destination node
- this also requires activities of each node to be in time order



```
uint32_t findIdOfRemoteDest( vector<SimActivity> &saque, SimActivity sa ){
    uint32_t min_lt = UINT_MAX;
    SimActivity sa_cand;
    uint32_t id_ret = sa.id;
    int ssize = saque.size();
    //printf("Activity ID %u : ConnectToRemote, lt(%u)=%u, type=%s\n", sa.id, sa.nid, sa.lt, type

    for( int i=first_id[sa.nid_dst]-1; i<ssize; i++ ){
        sa_cand = saque.at(i);
        //printf("sa.nid=%u, sa_cand : nid=%u id=%u lt(%u)=%u\n", sa.nid, sa_cand.nid, sa_cand.id
        if( (sa.id != sa_cand.id) && (sa_cand.nid == sa.nid_dst) && (sa.seq == sa_cand.seq) && (s
            //printf("%u connect to %u, s%u d%u ; s%u d%u, flag:%u %u ack: %u %u\n", sa.id, sa_ca
            id_ret = sa_cand.id;
            break;
        }
    }

    return id_ret;

}
```

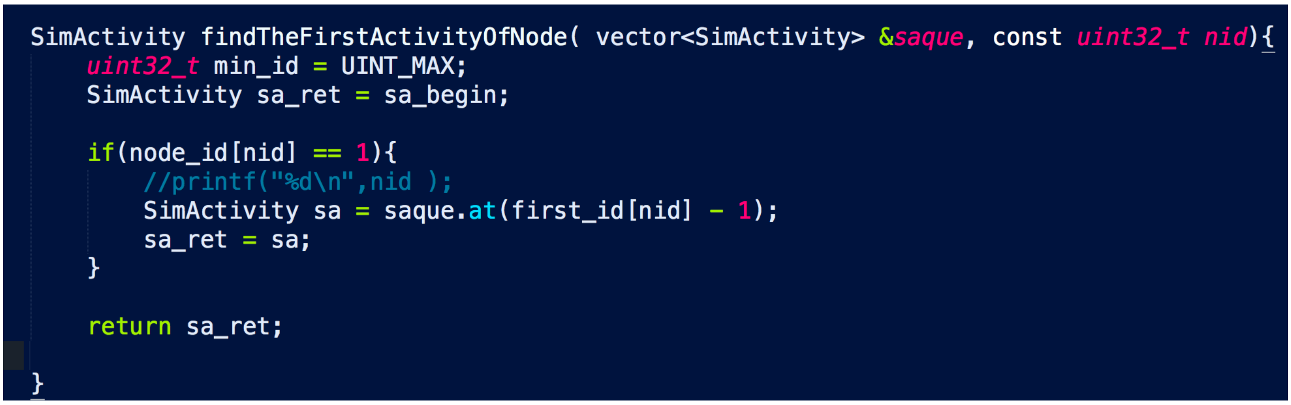
4. IN FUNCTION FIND_ID_OF_LOCAL_SRC

- actually, there is no need to find id of local source
- if the id belongs to the first activity of a node, the local source should be itself
- for other activities, the local source would be the previous activity

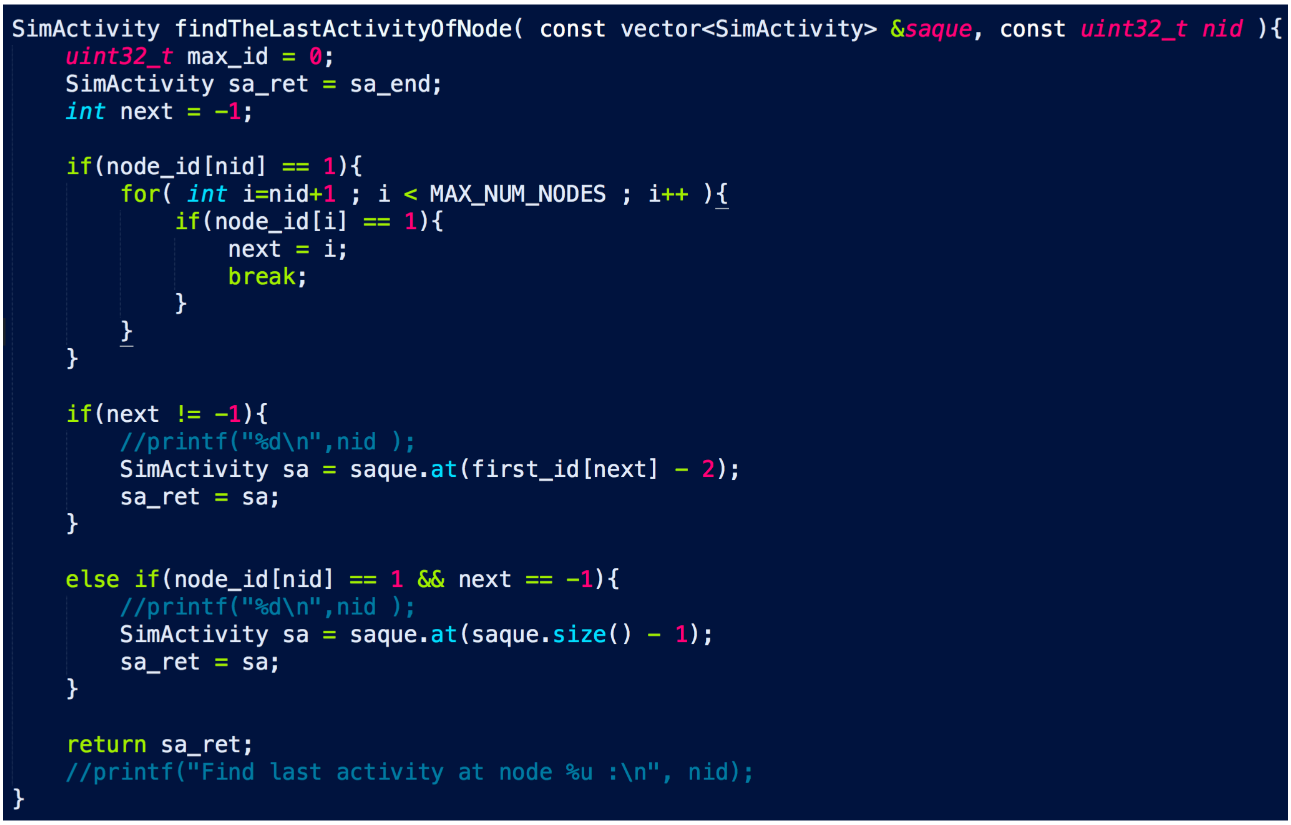


```
uint32_t findIdOfLocalSrc( vector<SimActivity> &saque, SimActivity sa ){
    int flag = 0;
    for(int n=0; n<MAX_NUM_NODES; n++)
    if(sa.id == first_id[n])
        flag = 1;
    if(flag == 0)
       return sa.id - 1;
    else
       return sa.id;

}
```

5. FIRST / LAST ACTIVITY OF NODES ARE ALREADY RECORDED



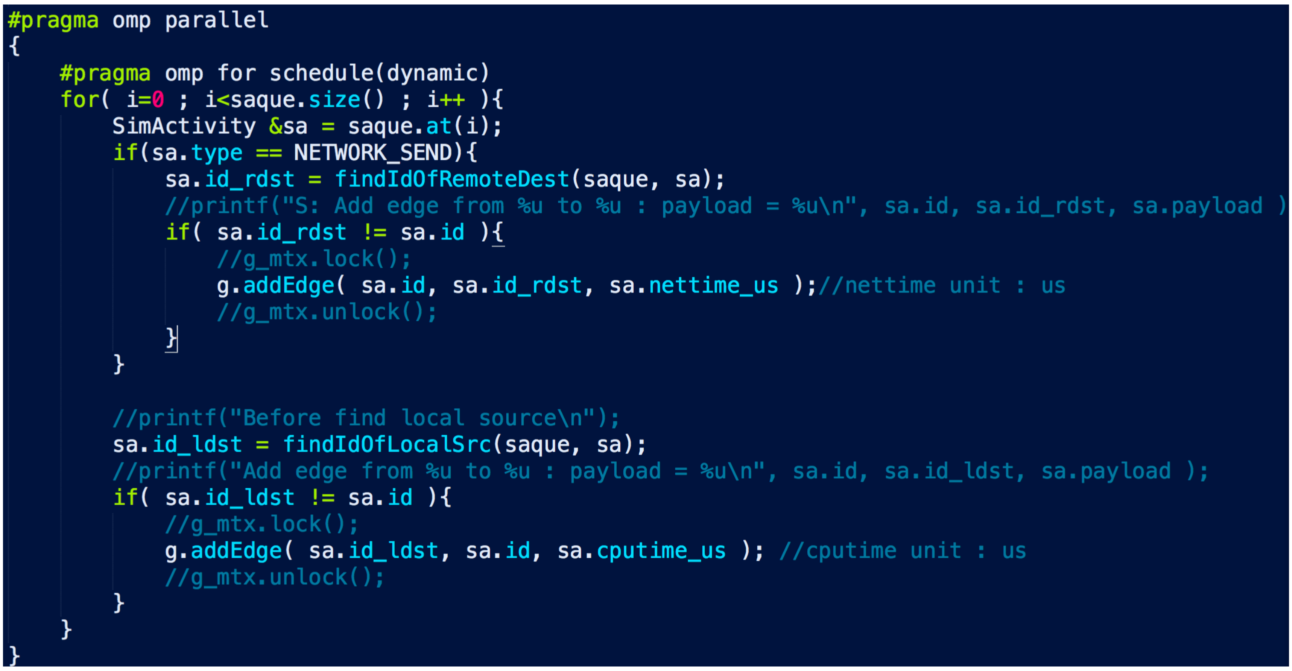
```
SimActivity findTheFirstActivityOfNode( vector<SimActivity> &saque, const uint32_t nid){
    uint32_t min_id = UINT_MAX;
    SimActivity sa_ret = sa_begin;

    if(node_id[nid] == 1){
        //printf("%d\n",nid );
        SimActivity sa = saque.at(first_id[nid] - 1);
        sa_ret = sa;
    }

    return sa_ret;

}
```



```
SimActivity findTheLastActivityOfNode( const vector<SimActivity> &saque, const uint32_t nid ){
    uint32_t max_id = 0;
    SimActivity sa_ret = sa_end;
    int next = -1;

    if(node_id[nid] == 1){
        for( int i=nid+1 ; i < MAX_NUM_NODES ; i++ ){
            if(node_id[i] == 1){
                next = i;
                break;
            }
        }
    }

    if(next != -1){
        //printf("%d\n",nid );
        SimActivity sa = saque.at(first_id[next] - 2);
        sa_ret = sa;
    }

    else if(node_id[nid] == 1 && next == -1){
        //printf("%d\n",nid );
        SimActivity sa = saque.at(saque.size() - 1);
        sa_ret = sa;
    }

    return sa_ret;
    //printf("Find last activity at node %u :\n", nid);
}
```

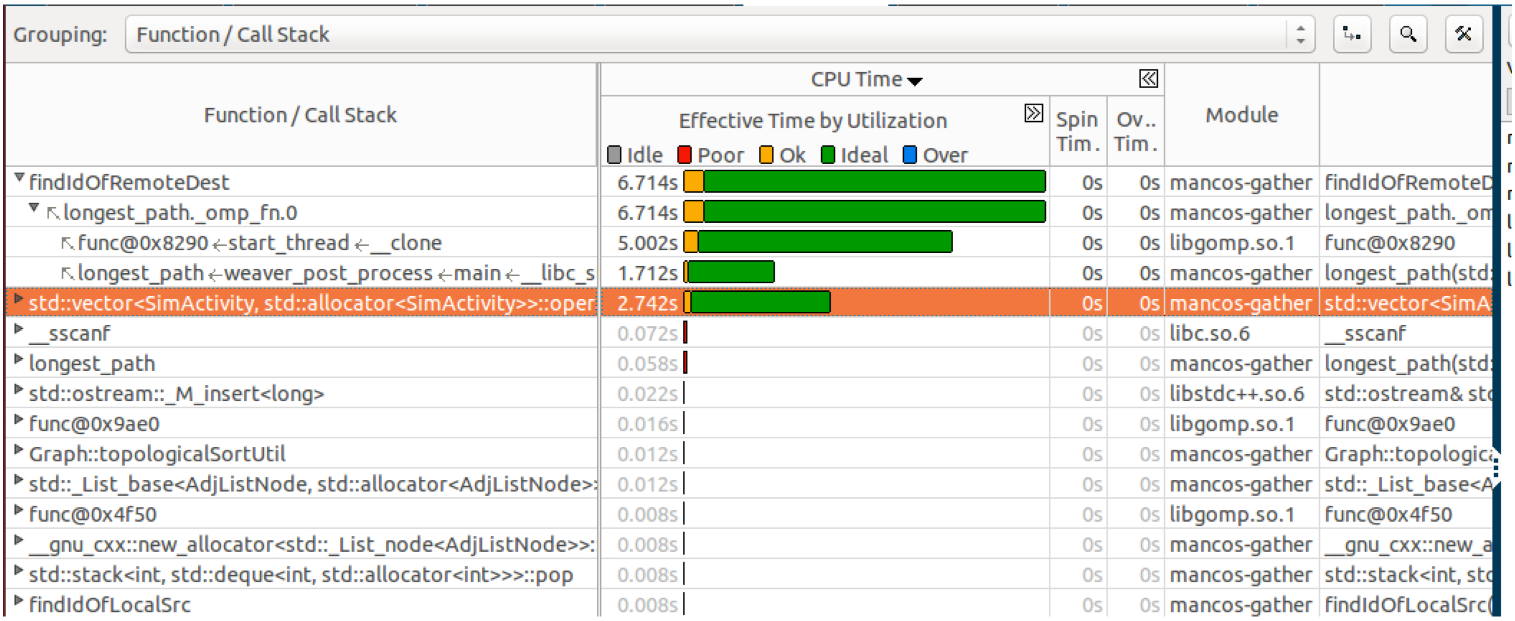
6. VECTOR INDEXING BY OPERATOR[] IS FASTER THAN OPERATOR .AT()

parallelize with OpenMP



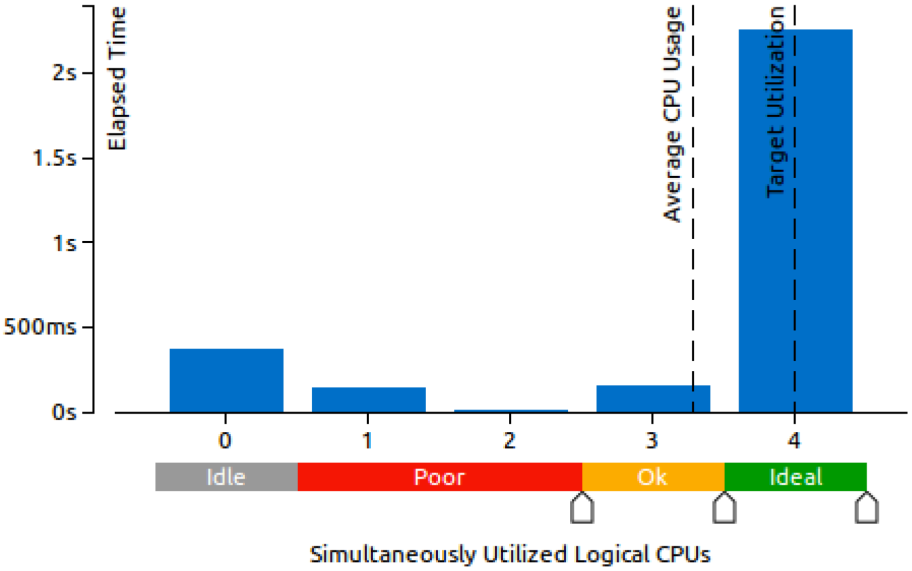
```
#pragma omp parallel
{
    #pragma omp for schedule(dynamic)
    for( i=0 ; i<saque.size() ; i++ ){
        SimActivity &sa = saque.at(i);
        if(sa.type == NETWORK_SEND){
            sa.id_rdst = findIdOfRemoteDest(saque, sa);
            //printf("S: Add edge from %u to %u : payload = %u\n", sa.id, sa.id_rdst, sa.payload )
            if( sa.id_rdst != sa.id ){
                //g_mtx.lock();
                g.addEdge( sa.id, sa.id_rdst, sa.nettime_us );//nettime unit : us
                //g_mtx.unlock();
            }
        }

        //printf("Before find local source\n");
        sa.id_ldst = findIdOfLocalSrc(saque, sa);
        //printf("Add edge from %u to %u : payload = %u\n", sa.id, sa.id_ldst, sa.payload );
        if( sa.id_ldst != sa.id ){
            //g_mtx.lock();
            g.addEdge( sa.id_ldst, sa.id, sa.cputime_us ); //cputime unit : us
            //g_mtx.unlock();
        }
    }
}
```

analysis of our results



Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization Idle Poor Ok Ideal Over	Spin Tim.	Ov.. Tim.	Module	
▼ findIdOfRemoteDest	6.714s	0s	0s	mancos-gather	findIdOfRemoteD
▼ ↖longest_path._omp_fn.0	6.714s	0s	0s	mancos-gather	longest_path._om
↖func@0x8290 ← start_thread ← __clone	5.002s	0s	0s	libgomp.so.1	func@0x8290
↖longest_path ← weaver_post_process ← main ← __libc_s	1.712s	0s	0s	mancos-gather	longest_path(std:
▶ std::vector<SimActivity, std::allocator<SimActivity>>::oper	2.742s	0s	0s	mancos-gather	std::vector<SimA
▶ __sscanf	0.072s	0s	0s	libc.so.6	__sscanf
▶ longest_path	0.058s	0s	0s	mancos-gather	longest_path(std:
▶ std::ostream::_M_insert<long>	0.022s	0s	0s	libstdc++.so.6	std::ostream& std
▶ func@0x9ae0	0.016s	0s	0s	libgomp.so.1	func@0x9ae0
▶ Graph::topologicalSortUtil	0.012s	0s	0s	mancos-gather	Graph::topologica
▶ std::_List_base<AdjListNode, std::allocator<AdjListNode>:	0.012s	0s	0s	mancos-gather	std::_List_base<A
▶ func@0x4f50	0.008s	0s	0s	libgomp.so.1	func@0x4f50
▶ __gnu_cxx::new_allocator<std::_List_node<AdjListNode>>:	0.008s	0s	0s	mancos-gather	__gnu_cxx::new_a
▶ std::stack<int, std::deque<int, std::allocator<int>>>::pop	0.008s	0s	0s	mancos-gather	std::stack<int, std
▶ findIdOfLocalSrc	0.008s	0s	0s	mancos-gather	findIdOfLocalSrc(

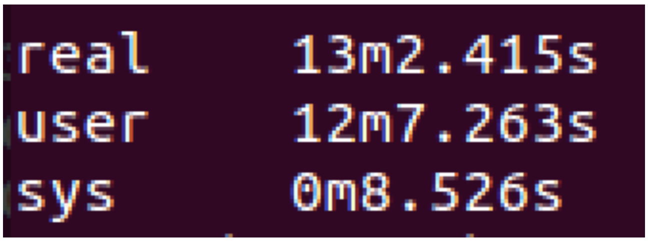


comparison

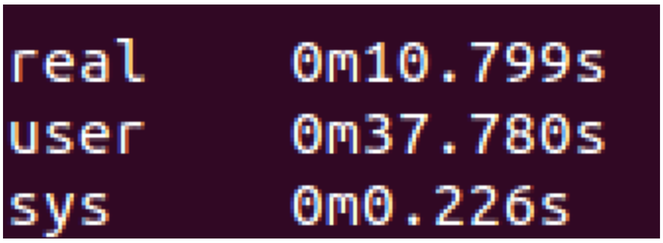
- CPU 2.4 GHZ INTEL CORE I5 TURBO UP TO 2.9GHZ
- 4 LOGICAL CPUS

ORIGINAL

OPTIMIZED & PARALLELIZED (4 THREADS)



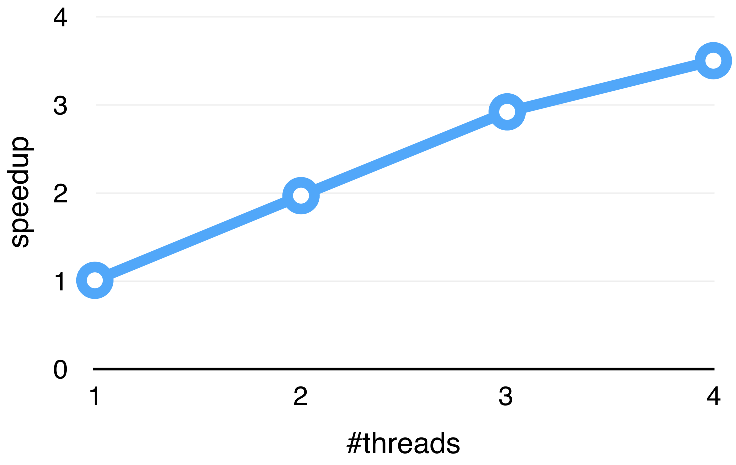
```
real        13m2.415s
user        12m7.263s
sys         0m8.526s
```



```
real        0m10.799s
user        0m37.780s
sys         0m0.226s
```

- **SPEEDUP**
  - total speedup about 72x
  - speedup due to multithreading 3.7/10 about 3.7
  - trying on different number of threads

	1 thread	2 thread	3 thread	4 thread
real time	26.419	13.931	11.159	10.799
user time	26.233	27.285	32.569	37.78
theoretical speedup	~1	1.96	2.91	3.49


speedup
#threads