**PHASE I**

1. Identify inner loops with only straight line code (i.e., no function calls etc)

These are target loops.

1. For the target loops build a DAG. Here nodes are operations and edges aee data dependecy.
2. At each node while creating DAG, assign scalar cost to it i.e.,op cost.
3. Topological sorting – to create levels and place nodes accordingly.

* Start at nodes with no incoming edges. (Levels 0 to max level, nodes at level 0 have no children)

**OUR MODIFICATIONS -**

* The phases are applied on each basic block separately so before the phaseI, decide which BBs are good enough to be vectorized based on their size i.e. no. Of instructions, and send only those to phase I.
* Vectorizing load/store operations.

Check best vectorizable set of ops in all the basic blocks, if aligned, vectorize their load/store ops. Else re-align & vectorize. (To be implemented later i.e., after the basic Dibyendu Das paper is implemented.)

Create a class named **node\_info** which contains various info related to a node.

***Class definition:***

class Node\_info

{

vector<value \*> //outgoing edges

llvm::Instruction \*pred 1; //incoming edges

llvm::Instruction \*pred 2; //incoming edges

unsigned opcost, vcost, scost, level, num;

};

class Node

{

llvm:: Instruction \*I;

Node\_info info;

};

typedef vector<Node> Nodes;

Nodes DAGNodes, levels[MAX];

Nodes::Iterator J = DAGNodes.begin();

for(; J != DAGNodes.end(); J++)

{

levels[J->second->level].insert(J);

}

//Find levels : starting from the 1st node, traverse 7 for each node assign the max depth

class def

{

unsigned node\_num;

string variable;

}

vector<def> deftable;

Datastructure for use-def chain :

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Instructions | Variable |
| I (n-1) | a |
| I(n-2) | z |
| . | . |
| . | . |
| . | . |
| I (k) | c |
| . | . |
| . | . |
| I(1) | b |