Nystrom inline support

1. Overview

The next generation points to analysis (nystrom) is a context sensitive (not finished yet), field sensitive, inclusion based points to analysis. For inter-proceduare nystrom points to analysis, its performed before inline. There is some drawback in current nystrom implementation in IPA, that lead to inaccurate points to info and alias relationship.

1. IPO_INLINE clone callee's local ST into caller's symtab, ConstraintGraph::cloneConstraintGraphMaps clone callee ST's constraint node to corresponding caller ST use ConstraintGraphNode::copy. The caller ST's points to set may conservertive for this call site. Because the original points to set is computed for all callsite.

2. Directly map other local constraint graph node (for example preg constraint graph node) into caller constraint graph in IPA_NystromAliasAnalyzer::updateCGForBE. caller_cg->_cgNodeToIdMap[callee_cg_node] = callee_cg_node->id, so the points to set still may conservertive for this inlined call site.

Following example to illusrate the problem

```
int a, b;
int c;
int foo1() {
   int *m;
   bar(&m, &a);
   *m += a;
   *m += b;
   return *m;
}

int foo2() {
   int *n;
   bar(&n, &b);
   *n += a;
   *n += b;
   return *n;
}

void bar(int **p, int *q){ // p points to m and n. q points to a and b.
  *q = c;
  *p = q;
}
```

After inline bar into foo1, with current nystrom inline maintenance, p and q's points to set in foo1 is same with p and q's points to set in bar. p points to m and n. q points to a and b. However after inilne, the accurate points to info for p and q is , p only points to m and q only points to a.

```
int foo1() {
   int *m;
   int **p;
   int *q;
   // inline_body_start pragma
   p = &m;
   q = &a;
   *q = c;
   *p = q;
   // inline_body_end pragma
   *m += a;
```

```
    *m += b;
    return *m;
}
```

To get better nystrom points to analysis result for such case, the solution is solving constraint graph node's points to set again when they are cloned into caller's constraint graph.

2. Nystrom inline support

The aim of this function is compute callsite specific points to info for constraint graph nodes that cloned from callee into caller.

Different with context sensitive nystrom analysis, which makes caller's constraint graph node obtain pointer analysis results equivalent to that obtained if a call graph were explicitly expanded. In above case, context sensitive nystrom analaysis can got the reuslt m only points to a. But still can't get q's accurate points to.

Conceptually, nystrom inline support takes two steps
1. Update Formal ST's points to set with actual's points to set, if formal ST's points to set is a strict subset of actual's points to set.
2. Begin with updated formal st's constraint graph nodes as working set, sovle the constraint graph node in callee. If node's points to set is updated with more accurate points to set, create a new node in caller's constraint graph record new points to set.

3. Design

nystrom analysis in IPO Inline.

```
IPO_INLINE::Process_Callee
    IPO_INLINE::Process_Formals
        IPO_INLINE::Process_Copy_In
            IPA_NystromAliasAnalyzer::processInlineFormal   // newly add in this patch
    .....
    ConstraintGraph::promoteLocals  // when inline happens, local static in callee is
    promoted to global, promote its constraint graph node from callee's constraint graph
    to global constraint graph.
    ConstraintGraph::cloneConstraintGraphMaps  // clone callee's local ST's constraint
    graph node into caller's constraint graph.
    IPA_NystromAliasAnalyzer::solveInlineConstraints  // newly add in this patch
    IPA_NystromAliasAnalyzer::updateCloneTreeWithCgnode // newly add in this patch
IPO_INLINE::Process_Callee end
```

data structure

```
class IPA_NystromAliasAnalyzer {
    ....
    // hashmap map original cg node id with cloned cg node for nystrom inline
    // key is callee constraint graph node's id.  value is optimized constraint graph
    node id in caller, this constraint graph node has moare accurate points to set in
    caller.
    hash_map<UINT32, UINT32> _inline_node_map;
}
```

IPA_NystromAliasAnalyzer::processInlineFormal(IPA_NODE *caller, IPA_NODE *callee, WN* actual, ST* formal_st)

1. Call callee_cg->checkCGNode(formal_st_idx, offset), get formal st's constraint graph node in callee cg (formal_node).
2. Call IPA_NystromAliasAnalyzer::getCGNode(actual, caller), get actual constraint graph node (actual_node).

```
    3. Union acutal node's all points to set (DN, GLB, ...), keep result in acutal_union.
    4. Check if actual_union is a strict subset of formal_node's down(DN) points to set.
    If so, this node can be optimized.
    5. Create a node in caller constraint graph (new_formal_node), it's DN points to set
    same as actual_union.
    6. Setup map between formal_node and new_formal_node in _inline_node_map.

IPA_NystromAliasAnalyzer::getCGNode(WN* wn, IPA_NODE* ipaNode)
    Get WN's constraint graph node in ipaNode's constraint graph.
    Not create new constraint graph node, when not find a constraint graph.

IPA_NystromAliasAnalyzer::solveInlineConstraints(IPA_NODE *callee, IPA_NODE *caller)
    1. Itearte _inline_node_map, find all formals updated with new constraint graph
    node. Add them into nodeWorkList.
    2. Iterate and udpate nodeWorkList, solve its out copyskew edge's dest node
    do {
      ConstraintGraphNode *old_node = nodeWorkList.pop();
      ConstraintGraphNode *new_node =
      ConstraintGraph::cgNode(getInlineNodeMap(old_node->id()));
      itearate old_node's out copyskew edges
         call IPA_NystromAliasAnalyzer::inlineSolveNode(edge->destNode(), callee_cg,
         caller)
         if return true which means edge->destNode's points to set is updated, add
         edge->destNode() into nodeWorkList.
    } while(!nodeWorkList.empty());

    Not handle out loadstore edges, because both ConstraintGraphSolve::processLoad and
    ConstraintGraphSolve::processStore add redundant copyskew edge for dest node.
    For example, formal node p points to x and y. p has a out load edge to constraint
    graph node q.
    Then ConstraintGraphSolve::processLoad will add copyskew edge from x and y to q.
    If after inline, we found p only points to x, but however there are copyskew edges
    from x and y to q. So q's points to is still x and y's points to set.

IPA_NystromAliasAnalyzer::inlineSolveNode(ConstraintGraphNode *node, ConstraintGraph*
callee_cg, IPA_NODE* caller)
    check if node's points to set can be more accurate. If so, update clone node with
    new points to set.
    1. Get node's point to set, call
    ConstraintGraphNode::UnionPointsToSet(origPointsTo), get union result for node's
    point to.
    2. Iterate node's incopyskew edges, exclude src_node's points to set from
    origPointsTo.
    3. If origPointsTo is not empty, this node can have more accurate points to set.
    4. Get node's corresponding clone node in caller constraint graph by
    IPA_NystromAliasAnalyzer::getCloneNode
    5. Clone_node exclude origPointsTo from its point to set.

IPA_NystromAliasAnalyzer::getCloneNode(ConstraintGraphNode *callee_node,
ConstraintGraph* callee_cg, IPA_NODE* caller)
    Get callee_node's corresponding node in caller's constraint graph.
    1. If callee_node alreay been cloned, mapped in _inline_node_map, return cloned node.
    2. Callee_node is local st node. Callee_node must be already cloned into caller's
    constraint graph by ConstraintGraph::cloneConstraintGraphMaps.
    3. Callee_node is preg node. create a new preg node in caller graph as clone node,
    copy callee_node's points to set to new preg node.
        Use preg StInfo's field _maxOffsets to record the max preg number used in its
```

constraint graph node.
For Var ST, _maxOffsets is used to hint how many constraint graph nodes allowed for this ST. If exceed this limit, all this ST's constraint graph nodes will be collapsed. Preg ST is never collapsed, so _maxOffsets field can be used to record max preg number.

4. Setup map between callee_node and clone_node in _inline_node_map.

IPA_NystromAliasAnalyzer::updateCloneTreeWithCgnode(WN* tree)
Iterate WN tree, if WN node has constraint graph node id, update its id with cloned node's id.

Another more complicated case is callee function inlined into same caller multiple times. In current IPO_INLINE implementation, when a callee function is inlined multiple times in same caller, its formal ST will mapped to serveral local ST in caller symtab. However its local ST will map to one local ST in caller symtab. In following example, after inline foo's symtab has two cloned ST for p, but only one cloned ST for q. The correct analysis steps for this case should be
1. After inline first call site, p1 points to a, q points to a.
2. After inline first call site, p2 points to b, q points to a and b.

```
foo() {
   int a, int b;
   call bar(&a)

   ...
   call bar(&b);
}

bar(int *p)
{
   int *q = p;
   ....
}
```

To handling such case, use two flags in nystrom inline supprot,
1. CG_NODE_FLAGS_VISITED, if node is solved and update points to set in IPA_NystromAliasAnalyzer::solveInlineConstraints.
2. CG_NODE_FLAGS_INLINE_NO_BENEFIT, if this node can't sovled when inline happens. For example, if constraint graph node is not sovled in first callsite, it's points to set same with nodes in callee function, then in later callsite, it's not worthwile to sovle in later callsite.

Modify ConstraintGraph::cloneConstraintGraphMaps
1. Clear CG_NODE_FLAGS_VISITED flags
2. If callee_node is cloned for first time.
Clear node's diff points to set.
3. If callee_node is alreay cloned and has no flag CG_NODE_FLAGS_INLINE_NO_BENEFIT.
This means callee function is already inlined once before this inline and this node is solved when inline last callsite.
Copy current points to set to diff points to set
(ConstraintGraphNode::deleteDiffPointsToSet).

ConstraintGraph::clearOrigToCloneStIdxMap(IPA_NODE *caller, IPA_NODE *callee)
1. Iterate all cloned nodes, cloned in ConstraintGraph::cloneConstraintGraphMaps
1. Node has no flag CG_NODE_FLAGS_VISITED, add flag CG_NODE_FLAGS_INLINE_NO_BENEFIT.
If node is solved when processing last callsite inline, copy orig_node's points to set to clone node.

2. Node has flag CG_NODE_FLAGS_VISITED, it's solved when processing current callsite inline.

Call ConstraintGraphNode::unionDiffToPts get union resutl for multiple inline.

2. Clear origToCloneStIdxMap.