Research Report

Thursday 12/05/2013

Wei Wang

review of the issues

- Test NLR with more programs with imperfect loop nests and complex loop boundaries
- Integrate NLR to PoCC (adding one more source code to the Collection)
- Addressing IMPACT 2014 final version issues
- Aim to get a paper (on what topic?) out in a CONFERENCE by March 1st, 2014.
- Energy Tuning Project
  - Lulesh benchmark using PPCG
  - PNNL benchmark

Progress & Problems

- Reconstruct loops and array accesses
  - Tested more imperfect loops (tested 2mm, 3mm in addition to gemm benchmark)
  - Triangle Loop Iterators (0<i<N; i<j<N) still fails (e.g. Covariance)
- Address IMPACT'14 reviewers' comments
  - Did not figure out why certain optimization caused power surge but still trying

The plan

- Make NLR work with triangle loop boundaries
- Integrate NLR to PoCC (adding one more source code to the Collection)
- Addressing IMPACT 2014 final version issues
- Aim to get a paper (on what topic?) out in a CONFERENCE by March 1st, 2014.
- Energy Tuning Project
  - Lulesh benchmark using PPCG
  - PNNL benchmark

Research Report

Monday 12/02/2013

Wei Wang

review of the issues

- Reconsturct loops and array accesses: issues 1) imperfect loop nest 2) complex boundary
- Test NLR on Windows Platform
- Integrate NLR to PoCC (adding one more source code to the Collection)
- Energy Tuning Project
  - Lulesh benchmark using PPCG
  - PNNL benchmark

Progress & Problems

- Reconstruct from the trace
  - NLR also works on imperfect loop nests (by setting parameters of NLR -- Professor Ketterlin's previous study addressed this aspect)
  - Tested Imperfect loop nests for gemm.c, success (see next slide)
  - NLR compiles and runs with Cygwin.

success case: gemm.c with imperfect loop nest

```
#pragma scop
  /* C := alpha*A*B + beta*C */
  for (i = 0; i < _PB_NI; i++)
    for (j = 0; j < _PB_NJ; j++)
      {
    C[i][j] *= beta;
    for (k = 0; k < _PB_NK; ++k)
      C[i][j] += alpha * A[i][k] * B[k][j];
      }
#pragma endscop
```

Above: original scop

>>>Right: constructed from trace (loop boundary set to 10), the code is correctly reconstructed.

```
#pragma scop
for ( i0 = 0; i0 <= 9; i0++)
    for ( i1 = 0; i1 <= 9; i1++)
        M[ 162668928 + 80*i0 + 8*i1 ] =
        M[ 162668928 + 80*i0 + 8*i1 ] +
        M[ 3214545984 ] ;
        for ( i2 = 0; i2 <= 9; i2++)
            M[ 162668928 + 80*i0 + 8*i1 ] =
            M[ 162668928 + 80*i0 + 8*i1 ] +
            M[ 3214545992 ] +
            M[ 162669760 + 80*i0 + 8*i2 ] +
            M[ 162670592 + 8*i1 + 80*i2 ] ;
#pragma endscop
```

The plan

- Test NLR with more programs with imperfect loop nests and complex loop boundaries
- Integrate NLR to PoCC (adding one more source code to the Collection)
- Addressing IMPACT 2014 final version issues
- Aim to get a paper (on what topic?) out in a CONFERENCE in 3 months (March 1st, 2014).
- Energy Tuning Project
  - Lulesh benchmark using PPCG
  - PNNL benchmark

Research Report

Monday 11/25/2013

Happy Holidays!

Wei Wang

review of the issues to solve/What I needed to do

- Needed to reconstruct loops and array accesses from the trace only
- Needed to write script to post process NLR outputs
- Needed to test with different programs (traces)
- Others
  - Lulesh (using INRIA PPCG)
  - PNNL benchmark

Progress & Problems

- Reconstruct from the trace
  - Looks like NLR works on *simple* (perfect) loop nest and *simple* loop iterators (lower =0, upper = N) but not (yet) imperfect loop or more complex loop bounds
  - gemver.c and gemm.c (perfect loop version) successfully constructed and analyzed
  - Imperfect loop nests (e.g. gemm.c and covariance.c) or complex loop bounds (covariance.c) failed
  - Example (both success and failed cases) in next slides

success program other than gemver.c (i.e. gemm.c)

```
#pragma scop
  /* C := alpha*A*B + beta*C */
  for (i = 0; i < _PB_NI; i++)
    for (j = 0; j < _PB_NJ; j++)
        C[i][j] = C[i][j]  * beta;
 for (i = 0; i < _PB_NI; i++)
    for (j = 0; j < _PB_NJ; j++)
        for (k = 0; k < _PB_NK; ++k)
        C[i][j] = C[i][j] + alpha * A[i][k] * B[k][j];
#pragma endscop

}
```

Above: original scop

>>>Right: constructed from trace (Loop Boundary set to 10), note that the constructed scop did not recover the first nested loop but the code is correct.

```
#pragma scop
for ( i0 = 0; i0 <= 99; i0++)
    M[ 146104704 + 8*i0 ] =
    M[ 146104704 + 8*i0 ] +
    M[ 3219847392 ] ;
for ( i0 = 0; i0 <= 9; i0++)
    for ( i1 = 0; i1 <= 9; i1++)
        for ( i2 = 0; i2 <= 9; i2++)
            M[ 146104704 + 80*i0 + 8*i1 ] =
            M[ 146104704 + 80*i0 + 8*i1 ] +
            M[ 3219847400 ] +
            M[ 146105536 + 80*i0 + 8*i2 ] +
            M[ 146106368 + 8*i1 + 80*i2 ] ;
#pragma endscop
~
```

failed program: gemm.c with imperfect loop nest

```
#pragma scop
  /* C := alpha*A*B + beta*C */
  for (i = 0; i < _PB_NI; i++)
    for (j = 0; j < _PB_NJ; j++)
      {
    C[i][j] *= beta;
    for (k = 0; k < _PB_NK; ++k)
      C[i][j] += alpha * A[i][k] * B[k][j];
      }
#pragma endscop
```

Above: original scop

>>>Right: constructed from trace (loop boundary set to 10), the code is not correctly reconstructed because the first 9 lines of code got repeated 100 times, where in reality it should be a loop (or a loop nest)

```
#pragma scop
M[ 162668928 ] =
M[ 162668928 ] +
M[ 3214545984 ] ;
for ( i0 = 0; i0 <= 9; i0++)
    M[ 162668928 ] =
    M[ 162668928 ] +
    M[ 3214545992 ] +
    M[ 162669760 + 8*i0 ] +
    M[ 162670592 + 80*i0 ] +
M[ 162668936 ] =
M[ 162668936 ] +
M[ 3214545984 ] ;
for ( i0 = 0; i0 <= 9; i0++)
    M[ 162668936 ] =
    M[ 162668936 ] +
    M[ 3214545992 ] +
    M[ 162669760 + 8*i0 ] +
    M[ 162670600 + 80*i0 ] +
```

failed program: covariance.c with complex loop bounds

Right: part of the original scop

```
/* Calculate the m * m covariance matrix. */
for (j1 = 0; j1 < _PB_M; j1++)
  for (j2 = j1; j2 < _PB_M; j2++)
    {
      symmat[j1][j2] = 0.0;
      for (i = 0; i < _PB_N; i++)
    symmat[j1][j2] += data[i][j1] * data[i][j2];
      symmat[j2][j1] = symmat[j1][j2];
    }
```

```
for ( i0 = 0; i0 <= 7; i0++)
  for ( i1 = 0; i1 <= 9 + -1*i0; i1++)
    val Write , 136754368 + 88*i0 + 80*i1 ,  , 136754368 + 88*i0 + 8*i1
val Write , 136755072 ,  , 136755072
val Write , 136755152 ,  , 136755080
val Write , 136755160 ,  , 136755160
```

Above: part of the reconstructed code. The original code is not correctly reconstructed from trace (loop boundary still set to 10). As you can see from above, the loop bounds are not correctly calculated.

The plan

- Discuss with Prof. Ketterlin about the two issues 1) imperfect loop nest 2) complex boundary
- Migrate NLR to Windows Platform
- Integrate NLR to PoCC (adding one more source code to the Collection)
- Any possibility of doing dependency analysis without candl?
- Energy Tuning Project
  - Lulesh benchmark using PPCG
  - PNNL benchmark

Research Report

Thursday 11/21/2013

Wei Wang

review of the issues to solve/What I needed to do

- Needed to know how existing programs perform the memory/dependency analysis
- Needed to reconstruct loops and array accesses from the trace only
- Others
  - Lulesh (using INRIA PPCG)
  - PNNL benchmark

Progress

- Reconstruct from the trace
  - With the help of Prof Ketterlin's Nested Loop Recognizer (NLR)
  - Change the format of NLR output to SCoP output (example shown in next slides)
  - Fake the operators because it does not affect dependency analysis!
  - Dump the SCoP with PoCC standard process flow: the SCoP (constructed from trace and NLR) ---> clan ----> candl

Generate SCoP from Trace (without knowing the source)

NLR, only one loop nest;

```
for i0 = 0 to 0x63
  for i1 = 0 to 0x63
    val Read
      , 0xbfb4ba04
      , Read
      , 0xbfb4ba08
      , Write
      , 0x804b060 + 800*i0 + 8*i1
      , Read
      , 0xbfb4ba04
      , Read
      , 0xbfb4ba08
      , Read
      , 0x804b060 + 800*i0 + 8*i1
      , Read
      , 0xbfb4ba04
      , Read
      , 0x8072480 + 8*i0
      , Read
      , 0xbfb4ba08
      , Read
      , 0x8072de0 + 8*i1
      , Read
      , 0xbfb4ba04
      , Read
      , 0x80727a0 + 8*i0
      , Read
      , 0xbfb4ba08
      , Read
      , 0x8072ac0 + 8*i1
```

Target SCoP Code

```
#pragma scop
 for ( i0 = 0 ;i0 < N;i++)
   for ( i1 = 0 ; i1< N;i++)
       A[i0][i1] =
     A[i0][i1] +
       B[i0] +
       C[i1] +
       D[i0] +
       E[i1] ;
 for ( i0 = 0 ;i0 < N;i++)
   for ( i1 = 0 ;i1 < N;i++)
       F[i0 ]  =
       F[i0 ] +
       G  +
       A[i0][i1]+
       H[i1];
 for ( i0 = 0 ;i0 < N;i++)
     F[i0]=
     F[i0]+
     I[i0];
 for ( i0 = 0 ;i0 < N;i++)
   for ( i1 = 0 ;i1 < N;i++)
       J[i0]  =
       J[i0]  +
       K  +
       A[i0][i1] +
       F[i1];
#pragma endscop
```

The Plan

- Write the program that does the previous conversion
  - Modify NLR code to dump out the required array form and loop form.
  - Some post processing to get rid of loop iterator read

Research Report

Monday 11/18/2013

Wei Wang

review of the issues to solve/What I needed to do

- Needed to know how existing programs perform the memory/dependency analysis
- Needed to know its weakness
- Needed to figure out where the memory trace could step in and help
- Others
  - Lulesh (using INRIA PPCG)
  - PNNL benchmark

Progress on PoCC Memory Trace

- Learned how SCoPs are represented by polyhedron in PoCC
- Looking into the *dependence polyhedron*
  - The code uses the *dependence matrix* to approximate (?) dependences
- Thought that eventually need to come up with a case containing pointers (within a scop?)
- Still No Signficant Progress
  - Still in the process of understanding how existing dependence analysis is done

The plan
(remain the same as previous two :( )

- Getting to know how existing programs perform the memory/dependency analysis
- Getting to know its weakness
- Think where the memory trace should step in and help

Research Report

Thursday 11/14/2013

Wei Wang

review of the issues to solve/What I needed to do

- Needed to know how existing programs perform the memory/dependency analysis
- Needed to know its weakness
- Needed to figure out where the memory trace could step in and help
- Others
  - Plos ONE (cardiac)
  - Lulesh (using INRIA PPCG)
  - PNNL benchmark

Progress on PoCC Memory Trace

- Thought that: the memory trace could at least be used to convince (or prove to) the compiler certain dependences (from static analysis) are false, given the trace collected dynamically.

- Still No substantial progress achieved
  - because got distracted from preparing for cardiac revised manuscript. Will submit it soon so that I would be focused before next report.

The plan
(remain the same as previous one)

- Getting to know how existing programs perform the memory/dependency analysis
- Getting to know its weakness
- Think where the memory trace should step in and help

Research Report

Monday 11/11/2013

Wei Wang

review of the issues to solve/What I needed to do

- Needed to understand the memory trace and let PoCC read in the trace and help memory analysis.
- Needed to work on a two pager describing the work I have been doing in the last 2 or 3 months (finished and sent)
- Others
  - Plos ONE (cardiac)
  - Lulesh (using INRIA PPCG)
  - PNNL benchmark

Progress on PoCC Memory Trace

- Ran the instrumented gemverOut.cpp
  - if N=4000, the output file becomes large (more than 2G)
  - Changed to smaller N for now
  - Tristan suggested the shape/dimension of the arrays should be considered as well
- Two related work from Clauss's group
  - Profiling Data-Dependence to Assist Parallelization: Framework, Scope, and Optimization (MICRO'12)
  - Online Dynamic Dependence Analysis for Speculative Polyhedral Parallelization (Euro-par'13)
  - Just got the idea, need to read in detail
- PoCC Code
  - Figured that Dependency Analysis should be the focus (CAnDL component of PoCC)
- No substancial progress achieved yet

The plan

- Getting to know how existing programs perform the memory/dependency analysis
- Getting to know its weakness
- Think where the memory trace should step in and help

Research Report

Thursday 11/07/2013

Wei Wang

review of the issues to solve/What I needed to do

- Needed to prepare and ship to you "one-icon click" version of Cygwin PoCC
- Needed to make MinGW work
- Others
  - Plos ONE (cardiac)
  - Lulesh (using INRIA PPCG)
  - PNNL benchmark

Progress on Windows PoCC

- Standalone Cygwin PoCC successfully prepared
  - This involved finding out what dlls and exe files (providing bash system on windows) are needed to enable a standalone version
  - Tested with different Windows platform on gemver.c (default test file)
- MinGW PoCC: not successful
  - Existing components of PoCC rely on POSIX compliant implementations offered by fork, wait, and pipe
  - MinGW, being minimalist GNU for Windows, just doesn't support the calls
  - PoCC has many places of calling posix functions like fork,pipe,and wait

The plan

- Focus on writing the 2-pager summarizing the work in the past 2-3 months.
  - Mini version of IMPACT submission
  - Brief mention of Windows version Polyhedral compiler
- Need also to work on PloS ONE re-submission (hopefully can put an end to the project)
  - Spend a few time on improving OpenACC performance on MIC (if not successful, should also be fine)
  - Text Edits

Research Report

Monday 11/04/2013

Wei Wang

review of the issues to solve/What I needed to do

- Needed to successfully build PoCC on CygWin
- Needed to build PoCC using mingw
- Whichever of the above two that provides the most easy way for others to try should be the focus
- Others
  - Plos ONE (cardiac)
  - Lulesh (using INRIA PPCG)
  - PNNL benchmark

Progress on Windows PoCC

- Cygwin successfully built
  - Update: it is possilbe to ship the cygwin dlls (only 4 involved) with PoCC.exe
  - Update: a little issue (though easy to solve) is PoCC.exe would call LINUX SCRIPTS to postprocess and assembel the output together
  - Planned action: also ship the SCRIPTS -- this needs a BASH system to also be shipped (bash.exe, find.exe, grep.exe from mingw can be a good choice)
- MinGW built
  - less library than Cygwin caused undefined variable
  - Still under investigation.
  - Set MinGW priority lower than making Cygwin-PoCC easy to try
  - After making Cygwin-PoCC work, immediately go back to this

The plan

- Cygwin "one-icon click" version ship to John soon
- MinGW-PoCC: try to make it work
- Others : depends on whether the above getting.

Research Report

Thursday 10/31/2013

Wei Wang

review of the issues to solve/What I needed to do

- Needed to make progress on Polyhedral compilers on Windows
- Needed to prepare some slides accompaning SC'13 UD Booth poster
- Others
  - Plos ONE (cardiac)
  - Lulesh (using INRIA PPCG)
  - PNNL benchmark

Windows Polyhedral Compilers

- No working version yet (Thursday 10/31)
  - 1. Cygwin Full Installation
  - 2. PoCC 1.2
  - 3. Fixed only two of three issues, when compiling PoCC in Cygwin (stumbling at the third issue)
    - Default Perl version in Cygwin too high, resolved by installing perl 5.10
    - two similar components have the same trivial variable array declared (resolved by manual renaming)
    - THIRD one: not able to resolve yet. Building clasttool components needed dynamic libraries (e.g. -lcloog-isl, -lpast), however PoCC compilation in Cygwin cannot generate dynamic libraries (dll?), native linux generated libcloog-isl.so and libpast.so

Others

- Prepared slides for SC13 UD Booth that go with the poster
- Others: no progress except cleaned up all results relating to IMPACT submission

The plan

- Figure out how to generate .so counter part in Cygwin for components like cloog, past etc.
- Try to compile component by component
- Learn how to develop DLLs in Windows Environment and apply to all components of PoCC?
- Turn to PPCG and use Visual Studio 2013?
- Need to work for the resubmission of PlosONE (30 days passed, recommended 45 days resubmission)

Research Report

Monday 10/28/2013

Wei Wang

review of the issues to solve/What I needed to do

- Short: submit the draft to Impact 2014
- Medium: submit plos one by November
  - This needs making the cardiac code more efficient
- Long: think about what should I do to extend the current impact 2014 draft to aim for a CONFERENCE paper and for my proposal

What progress did I make

- Submitted IMPACT 2014
- Prepared the poster for SC'13. The contents were all from IMPACT 2014 submitted paper

The Plan

- Cardiac Project
  - Work on getting more speedups on MIC (with the help of VTune)
  - Work on modifying the text
  - Get a ready version by next Monday and send out
- Energy Profiling
  - Prepare required slides for SC'13, accompanying the poster.
  - Start looking into the set of PNNL benchmarks

Research Report

Thursday 10/24/2013

Wei Wang

review of the issues to solve/What I needed to do

- IMPACT 2014 workshop on polyhedral compilation techniques due on 10/25/2013 AoE, needed to 1) finish related work and conclusion part 2) collaborate with John and Allan with the draft
- Not sure whether to include not-so-good Lulesh results to impact 2014 paper
- Manual implementation of cardiac code (for Plos ONE submission) has too much thread wait for OpenMP implementation

progress on Impact Workshop 2014

- Finished related work and conclusion before last Tuesday
- Worked with Allan improving the draft on Tuesday, Wednesday, and Today
- Thought that we better include lulesh results, even though it didn't get speedups. (at least it can support Time-Energy Correlation)
- Need to go through the draft and get it submitted before Sunday 8AM!

progress on lulesh

- Generated and ran 200 program variants with 100 maxfuse + 100 smart fuse (with different tiling size)
- Need to collect the results and add the resulting graph to the draft

progress on Plos ONE

- No progress from Monday
- Will dedicate next week to Plos ONE and aim to submit it before November.

The plan

- Short: submit the draft to Impact 2014
- Medium: submit plos one by November
  - This needs making the cardiac code more efficient
- Long: think about what should I do to extend the current impact 2014 draft to aim for a CONFERENCE paper and for my proposal....!!??

Weekly Research Report

Monday 10/21/2013

Wei Wang

review of the issues to solve/What I needed to do

- Although Lulesh can go through polyhedral compilers, no extensive experiments were done, no results (numbers /graphs) were generated
- IMPACT 2014 workshop on polyhedral compilation techniques due on 10/25/2013 AoE, need to 1) get lulesh results into paper & 2) get a ready draft to John by Monday.
- Manual implementation of cardiac code (for Plos ONE submission) is not well vectorized

progress on: IMPACT 2014 draft

- Added MIC results of Cardiac code to the draft
- Redo all parts of the draft except Related Work and Conclusion
- Sent the draft to John
- Need to work on related work and conclusion
- Depend on how many pages we want, we can add some discussion of lulesh challenges

progress on: Lulesh

- Generated program variants for one SCoP
  - that scop occupied ~12% execution in fully parallelized OpenMP
  - did not generate variants for more SCoP because the polyhedral compiler took too long to do transformation
  - The results on elo is that the variants performed as well as the original OpenMP implementation on Sandy Bridge
  - The original OpenMP program had slow down running on MIC.
  - Not planning to add lulesh results to IMPACT2014 draft

progress on: PlosONE

- Met with Will on profiling the cardiac code using Intel Vtune Amplifier
  - we found that the vectorization is at the OK range.
  - The problem is related to memory stalls and also the bandwidth is not 100% utilized.

The plan (before Thursday)

- Related work and conclusion section of IMPACT 2014 workshop paper
- Try to get good Lulesh numbers using polyhedral approach still
  - at least I will have the energy and time correlation data generated on Sandy Bridge (on MIC, the problem is aforementioned: the program is not performance portable yet: needs investigation)

Weekly Research Report

Thursday 10/17/2013

Wei Wang

What I needed to do (review)

- work on using PPCG compiler to transform Lulesh programs
  - worked on it (details in next slides)
- work on the vectorization of the cardiac code
  - Did not work on it yet.

What did I do (1)

- work on using PPCG compiler to transform Lulesh programs
  - Riyadh's PPCG could auto-parallelize (involving scalar privatization) the sequential code while tiling it. But the problem is: the time it took to auto-parallelize is at the magnitude of hours!
  - Solution: ?

What did I do (2)

- Polyhedral optimize cardiac code for input size smaller than 2048, on Xeon Phi
  - Previously, input size 2048 had barely speedups (about 3-5% improvement)
  - Found that for smaller input size, the improvement was about 15%. For example, size 1024, the 110X speedup got improved to 110*(1+15%) = 130X speedup by using polyhedral transformations. (Graphs attached in the beginning)

What do I plan to do?

- Get lulesh auto-parallelized and transformed for MIC, put results into paper draft
- Get a ready to submit version to you by Sunday (10/20/2013) for IMPACT 2014 workshop on polyhedral compilation techniques (deadline 10/25/2013 AoE)
- Get some progress on improving manual implementation of cardiac code.
- Start looking at Windows PoCC/PPCG