TRACO manual

Faculty of Computer Science
West Pomeranian University of Technology in Szczecin, Poland
`traco.sourceforge.net`

March 19, 2013

1 Preface

TRACO is a source-to-source parallelizing compiler. It transforms program loops written in C/C++ language. The output code is compilable. The parallelism is described using the OpenMP directives. TRACO automatically searches loops in input code and replace them for appropriate ones.

The algorithms of the TRACO tool are written in C/C++ and are explained in papers [1, 2]. TRACO allows users to extract synchronization-free slices, calculate free-schedule, extract parallelism in inner loops, and use variable privatization and reduction.

TRACO includes the dependence analyser Petit [8] and Omega Calculator for Pressburger arithmetic operations [3]. TRACO can use also optionally libraries isl and CLOOG [4] instead of Omega codegen function. Source-to-source transformations are written in Python.

2 Installation

The environment of the TRACO is any distribution of the LINUX operating system. The recommended way to get the fresh sources (with updates) of TRACO is SVN repository.

```
svn checkout svn://svn.code.sf.net/p/issf/code-0/trunk issf-code-0
```

TRACO can be also downloaded from the website `http://sourceforge.net/projects/issf/`, however this stable package is usually older than sources in the SVN repository.

Next, the sources have to be compiled. Some additional libraries are required. Most of them are part of the Linux distribution.

```
flex
bison
boost-graph
boost-regex
python
python-pip
realpath
libgv-python
pyparsing-python
```

The additional packages for Python are also required. It can be installed as:

```
pip install easyprocess
pip install python-graph-core
pip install python-graph-dot
```

The following command compiles the TRACO tool.

```
make all
```

To use CLOOG [4] and isl [5] the barvinok package is required. Barvinok can be downloaded from the website `www.kotnet.org/~skimo/barvinok`. The file: *barv_script* in TRACO sources keeps path to the barvinok tool (iscc). Please change for appropriate one. The package is recommended.

The package Omega+ is optional and can be found at the website `http://www.cs.utah.edu/~chunchen/`. The file: *omegaplus_script* in TRACO sources keeps path to the Omega+ tool (oc). Please change for appropriate one.

The following command allows for usage of the TRACO without path (root privileges are required):

```
ln -s path_to/issf /usr/bin/issf
```

3 Usage

TRACO is a command line tool - **issf**.

The following command parallelizes test.c program:

```
issf test.c
>> 5 loops found.
>> 5 loops PARALLELIZED.
>> test_par.c generated.
```

The output is written to a *namefile*_**par**.*extenstion* file. TRACO prints how many program loops have been found and parallelized. The example listings of input and output programs are in appendix A.

issf can be run with the following flags:

- –tc - chooses the algorithm for transitive closure [0,1,2]. 0 - Omega algorithm [3], 1,2 - algoritms presented in [6].
- –codegen - the algorithm for loop code generator [0,1,2]. 0 - Omega algorithm, 1 - CLOOG algorithm, 2 - Omega+ algorithm.
- –scc - divide loop for SCC loops by means of mutual-accessibility (strongly connected components).

- –tiling - tile loop (an experimental novelty)
- –silent - enables more informations [0,1] in the tmp/debug.txt when 0 (slower processing), no debug when 1.

Example:

```
issf test.c --tc=1 --codegen=1
```

4 For developers

TRACO creates tmp folder. In this folder you can find deps.txt files with dependence relations of program loops; debug.txt with information for developers; petit, pseudocode and compilable code of invidual program loops. Source-to-source transformation systems are realized by Python scripts localized in the *py* folder of the TRACO sources.

It is possbile to use the core of the ISS Framework directly which only generates pseudocode and requires program loops in Petit syntax.

```
framework/obj/framework petit_loop.t
```

It allows developers to debug and observe relations calculations and raw data from framework. Flags are explainded in the framework/framework.c file.

To generate CUDA/OpenCL code please setup the Cetus OpenMP2GPU driver [7] and transform the output OpenMP program from the Traco tool.

```
java -classpath ./cetus.jar:./antlr.jar omp2gpu.exec.OMP2GPUDriver test_par.c
```

ANTLR and JAVA software is required. In detail, usage of the Cetus is explained at the website: `http://cetus.ecn.purdue.edu/`.

5 Support and Future Work

Please send information about your experience with the Traco compiler to the address: `marekp1980@gmail.com`.

It allows us to improve the tool. TRACO is being developed tool and can contain some bugs or setup difficulty (specially in SNV sources). We keep up to date with fixes.

In the future we plan to add a locality optimize engine for TRACO by means of multi-dimensional tiling.

6 Citations

```
@article{BieleckiPK12,
  author    = {Wlodzimierz Bielecki and
               Marek Palkowski and
               Tomasz Klimek},
  title     = {Free scheduling for statement instances of parameterized
               arbitrarily nested affine loops},
  journal   = {Parallel Computing},
  volume    = {38},
  number    = {9},
  year      = {2012},
  pages     = {518-532},
  ee        = {http://dx.doi.org/10.1016/j.parco.2012.06.001},
}


@article{BeletskaBCPS11,
  author    = {Anna Beletska and
               Wlodzimierz Bielecki and
               Albert Cohen and
               Marek Palkowski and
               Krzysztof Siedlecki},
  title     = {Coarse-grained loop parallelization: Iteration Space Slicing
               vs affine transformations},
  journal   = {Parallel Computing},
  volume    = {37},
  number    = {8},
  year      = {2011},
  pages     = {479-497},
  ee        = {http://dx.doi.org/10.1016/j.parco.2010.12.005},
}
```

7 Appendix A

test.c

```
#include <stdio.h>
#define N 100

int main()
{
   int a[101], i, j, b[101][101],c;
   for(i=1; i<=100; i++){
      j = j * a[i];
   }

   for(i=1; i<N; i++){
      for(j=1; j<=100; j++){
              c = 2;
         b[i][j] = b[i][j-1] + c;
      }
   }

   for ( l=1 ; l<=loop ; l++ ) {
      for ( k=0 ; k<n ; k++ ) {
              x[k] = q + y[k]*( r*z[k+10] + t*z[k+11] );
      }
   }

   for ( l=1 ; l<=loop ; l++ ) {
      for ( i=0 ; i<25 ; i++ ) {
         for ( j=0 ; j<n ; j++ ) {
            px[j][i] += vy[k][i] * cx[j][k];
         }
      }
   }

   for(i=1; i<N; i++){
      for(j=1; j<=100; j++){
         b[i][j] = b[i][j+1] + b[i+1][j];
      }
   }


   return 0;
}
```

test_par.c

```
int i,j,t2,t1,t3;

#include <omp.h>
#include <stdio.h>
#define N 100

int main()
{
   int a[101], i, j, b[101][101],c;

#pragma omp parallel for private(i) reduction(* : j)
   for(i=1; i<=100; i++){
        j = j * a[i];
   }

#pragma omp parallel for private(i,j,c)
   for(i=1; i<N; i++){
          for(j=1; j<=100; j++){
                  c = 2;
             b[i][j] = b[i][j-1] + c;
        }
   }

if (loop >= 2) {
  #pragma omp parallel for  private(t2,t1)
  for(t2 = 0; t2 <= n-1; t2++) {
      for(t1 = 1; t1 <= loop; t1++) {
        x[t2]=q+y[t2]*(r*z[t2+10]+t*z[t2+11]);
      }
  }
}


if (loop >= 2 && n >= 1) {
  #pragma omp parallel for  private(t2,t3,t1)
  for(t2 = 0; t2 <= 24; t2++) {
    for(t3 = 0; t3 <= n-1; t3++) {
      if (loop >= 2 && t3 >= 0 && n >= t3+1 && t2 >= 0 && t2 <= 24) {
        for(t1 = 1; t1 <= loop; t1++) {
          px[t3][t2]+=vy[k][t2]*cx[t3][k];
        }
      }
    }
  }
}

if (N >= 2) {
  b[1][1]=b[1][1+1]+b[1+1][1];
}
if (N >= 2) {
  for(t1 = 1; t1 <= N+97; t1++) {
    #pragma omp parallel for private(t2)
    for(t2 = max(t1-98,1); t2 <= min(N-1,t1+1); t2++) {
      b[t2][t1-t2+2]=b[t2][t1-t2+2+1]+b[t2+1][t1-t2+2];
    }
  }
}


   return 0;
}
```

Bibliography

[1] Beletska, A., Bielecki, W., Cohen, A., Palkowski, M., Siedlecki, K., : Coarse-grained loop parallelization: Iteration space slicing vs affine transformations. Parallel Computing, 37, pp. 479-497, (2011).

[2] Bielecki, W., Palkowski, M., Klimek, T., Free scheduling for statement instances of parametrized arbitrarily nested affine loops, Parallel Computing, Volume 38 Issue 9, 518-532, (2012).

[3] Wonnacott, D., : A Retrospective of the Omega Project, Haveford College Computer Science Tech Report 01-2010, (2010).

[4] Bastoul, C., : Code Generation in the Polyhedral Model Is Easier Than You Think, PACT'13 IEEE International Conference on Parallel Architecture and Compilation Technique, Juan-les-Pins, France, pp.7-16, (2004).

[5] Verdoolaege, S., Integer Set Library - Manual, `www.kotnet.org/~skimo/isl/manual.pdf`, (2011).

[6] Bielecki, W., Klimek, T., Palkowski, M., Beletska, A., : An Iterative Algorithm of Computing the Transitive Closure of a Union of Parameterized Affine Integer Tuple Relations, Lecture Notes in Computer Science, vol. 6508/2010, pp. 104–113, (2010).

[7] Chirag, D., et al., : Cetus: A Source-to-Source Compiler Infrastructure for Multicores, IEEE Computer, pp. 36-42, (2009).

[8] Kelly, W., Pugh, W., Maslov, V., Rosser, E., Shpeisman, T., Wonnacott, D., : New User Interface for Petit and Other Extensions. User Guide, (1996).