Automated parallelization of sequential C-programs on the example of two applications from the field of laser material processing

M.S. Baranov[12], D.I. Ivanov[12], N.A. Kataev[1], A.A. Smirnov[1]

Keldysh Institute of Applied Mathematics Russian Academy of Sciences[1]
Lomonosov Moscow State University[2]

Moscow, September 28-29, 2015

The proposed approach to a semi-automatic parallelization

The optimization and parallelization of programs are an execution of a sequence analysis and transform passes. The choice of the optimal sequence depends on the application, the purpose of optimization, the architecture of the target computer system and technologies used for parallel programming.

Who chooses the sequence of these passes?

User describes a transform pass that should be performed.

Who performs each pass?

The described passes are performed in automatic way.

Implementation

1. There are a number of basic transformations that can be performed in an automatic way: variable propagation, loop-invariant code motion, loop unrolling, loop merging, loop distribution, etc.
2. User selects basic transformations that should be performed and annotates a program.
3. Before the execution of transformations the assistance tool checks their permissibility.

Example of a basic transformation: loop unrolling

Problem

The pattern of access to elements of the arrays Flx and Flz on different iterations is unknown. Loop-carried dependencies may arise.

Solution

Simplify these accesses using loop unrolling transformation.

```
for (I = 1; I < Nx + 1; ++I)
  for (J = 1; J < Nz + 1; ++J)
    #pragma x unroll
    for (II = 0; II < 2; ++II)
      for (JJ = 0; JJ < 2; ++JJ) {
        …
        Flx[I][J - 1 + JJ] = Flx[I][J - 1 + JJ] - Iloc[0] * Fl[0];
        Flz[I - 1 + II][J] = Flz[I - 1 + II][J] - Jloc[1] * Fl[1];
      }
```

⇨

```
for (I = 1; I < Nx + 1; ++I)
  for (J = 1; J < Nz + 1; ++J) {
    …
    Flx[I][J - 1] = Flx[I][J - 1] - 1 * Fl[0];
    Flz[I - 1][J] = Flz[I - 1][J] - 1 * Fl[1];
    …
    Flx[I][J] = Flx[I][J] - 1 * Fl[0];
    Flz[I - 1][J] = Flz[I - 1][J] - -1 * Fl[1];
    …
    Flx[I][J - 1] = Flx[I][J - 1] - -1 * Fl[0];
    Flz[I][J] = Flz[I][J] - 1 * Fl[1];
    …
    Flx[I][J] = Flx[I][J] - -1 * Fl[0];
    Flz[I][J] = Flz[I][J] - -1 * Fl[1];
}
```

#pragma x unroll [(index-list)]

index ::= var : int-value

Semi-automatic parallelization of applications Powder 2D and Powder 3D

These applications implement the two-dimensional and three-dimensional problems of modeling of melting of multi-component powder in selective laser sintering based on multicomponent and multiphase hydrodynamic model.

The programs have been parallelized using parallel programming technologies OpenMP and OpenACC.

x 7

on 8 threads
= 3,15 x 2,2

	Original	Transformed	Auto (original)	Auto (transformed)	OpenMP	OpenACC
1 thread	50,59	14,19			16,06	81,12
2 threads					10,76	
4 threads					8,26	
8 threads			62,8	9,9	7,19	
1 GPU						9,37

Hardware
- CPU: 4-cores processor Intel Core i7-3770 CPU 3.40GHz with active Hyper Threading (2 threads per core);
- GPU: NVIDIA GTX Titan (Kepler generation).

Compilers
- Intel C Compiler V15.0 with option -O3 for sequential and OpenMP programs. Results of automatic parallelization with option -parallel are also presented at the table above.
- PGI V15.1-0 with option –O3 for OpenACC programs.

Thank you for attention ☺

?

http://dvm-system.org/