SCIENTIFIC CONFERENCE
Russian Supercomputing Days

LLVM Based Parallelization of C Programs for GPU

Nikita Kataev

Keldysh Institute of Applied Mathematics RAS, Moscow, Russia

DVM SYSTEM

September 22, 2020 | Moscow

Goals of SAPFOR

SAPFOR (System For Automate Parallelization) is a software development suit that is focused on cost reduction of manual program parallelization.

The main goals of SAPFOR development:

- Exploration of sequential programs (program analysis and profiling).
- Automatic parallelization (according to DVMH or OpenMP models) of a well-formed program for which a programmer maximizes algorithm-level parallelism and asserts high-level properties (implicit parallel programming methodology).
- Semi-automatic program transformation to obtain a well-formed sequential version of the original program.

DVMH Programming Model

Development of high-level parallel programming languages.
Directive-based programming models.

CUDA

OpenMP

MPI

DVMH

Directive-based programming model which aims to create parallel programs for heterogeneous computational clusters (GPU NVidia, Intel Xeon Phi, multicore CPUs).

The model includes two programming languages which are the extensions of standard C and Fortran languages by parallelism specifications: *CDVMH* and *Fortran-DVMH*

The parallel program is developed in terms of a sequential one.

DVMH Programming Model

Development of high-level parallel programming languages.

CUDA

Ope

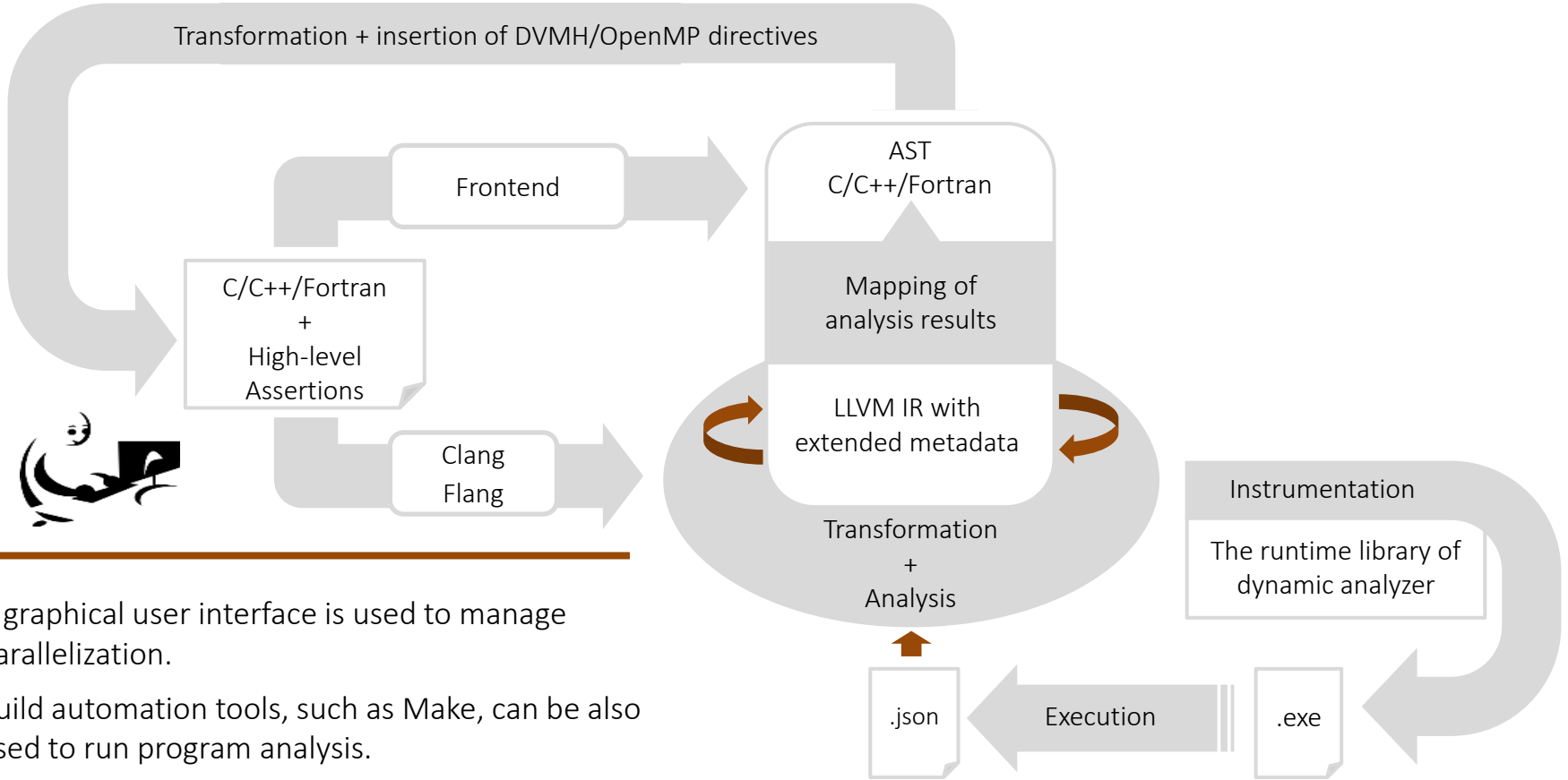
MPI

```
float A[N][N];
float eps = 0.f;

...
#pragma dvm actual(eps)

#pragma dvm region
{
#pragma dvm parallel([i][j]) tie(A[i][j]) across(A[1:1][1:1] \
                                                 reduction(max(eps))
for (i = 1; i < N - 1; i++)
  for (j = 1; j < N - 1; j++)
  {
    float s = A[i][j];
    A[i][j] = (w / 4) * (A[i-1][j] + A[i][j-1] + A[i][j+1] + A[i+1][j])
            + (1 - w) * A[i][j];
    eps = Max(fabs(s - A[i][j]), eps);
  }
}
#pragma dvm get_actual(eps)
```

Architecture of SAPFOR



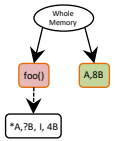
- A graphical user interface is used to manage parallelization.
- Build automation tools, such as Make, can be also used to run program analysis.

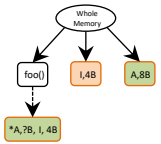
Program analysis and transformation

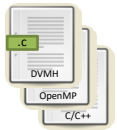
Produce AST + LLVM IR with Clang

Construct source-level alias tree

Use metadata to store its nodes in LLVM IR







Transformation of the source code

Analysis passes

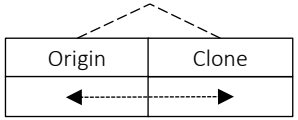
Parallelization pass

Request for analysis results from server, explore original LLVM IR to determine parallelization opportunities of the original program

Analysis servers (separate threads)

Clone LLVM module

Alias tree handles allow us to track memory locations across transformation of LLVM IR



Bidirectional map stores memory locations.

Rebuild the alias tree.

Correspondence with original alias tree is established and the bidirectional map is initialized.

Transform and analysis passes

Automatic Parallelization Using DVMH Model

Parallelization for compute devices with shared memory (multi-core CPU or accelerator) requires that three kinds of annotations be inserted into the source code:

- specifications of the loops which can be executed in parallel, as well as specifications of private and reduction variables,
- specification of the compute regions which can be executed on the accelerators, each region may enclose one or more parallel loops,
- high-level specifications of data transfer between a memory of CPU and a memory of accelerator (actualization directives).

For each loop the following constraints are examined:

- safety of control flow (the absence of I/O operations, side effects, etc.),
- safety of memory accesses (absence of loop-carried data dependencies and captured pointers),
- the direction of data usage (input, output, and local data),
- canonical loop form according to the OpenMP standard,
- the ability to express properties of memory locations with DVMH directives,
- the ability to collapse iteration spaces of nested loops into one larger iteration space.

Optimization of CPU-to-GPU Data Transfer

We want to prepare data for the accelerator as early as possible and to request data from the accelerator as late as possible.

Explore the direction of data usage for each loop.

```
for (…) {
   K = …     // local
   … = B[I] // in
   A[K] = … // out
}
```

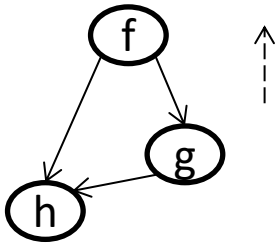
⇨

```
#pragma actual(B)
  for (…) {…}
#pragma get_actual(A)
```

1

We use postorder traversal to reduce the amount of data transfer.

```
h() -> g() -> f()
```



2

Optimization of CPU-to-GPU Data Transfer

Join actualization directives for neighboring regions.

```
#pragma actual(B)
  for (…) {…}
#pragma get_actual(A)
#pragma actual(B)
  for (…) {…}
#pragma get_actual(A)
```

⇨

```
#pragma actual(B)
  for (…) {…}
  for (…) {…}
#pragma get_actual(A)
```

2.a

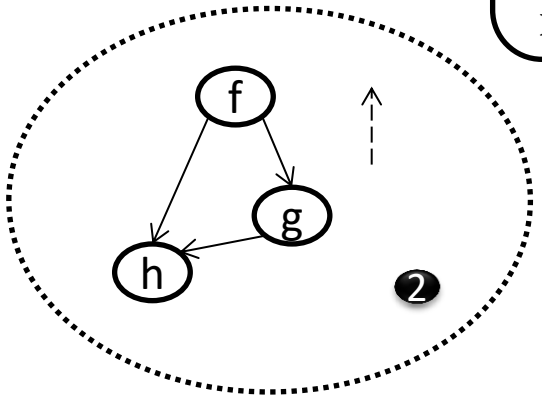
Move actualization directives outside the body of a sequential loop.

```
for (…) {
#pragma actual(B)
#pragma parallel (1)
  for (…) {…}
#pragma get_actual(A)
}
```

⇨

```
#pragma actual(B)
for (…) {
  for (…) {…}
}
#pragma get_actual(A)
```

2.b



Move actualization directives outside the callees.

```
foo();

void foo() {
#pragma actual(B)
  …
#pragma get_actual(A)
}
```

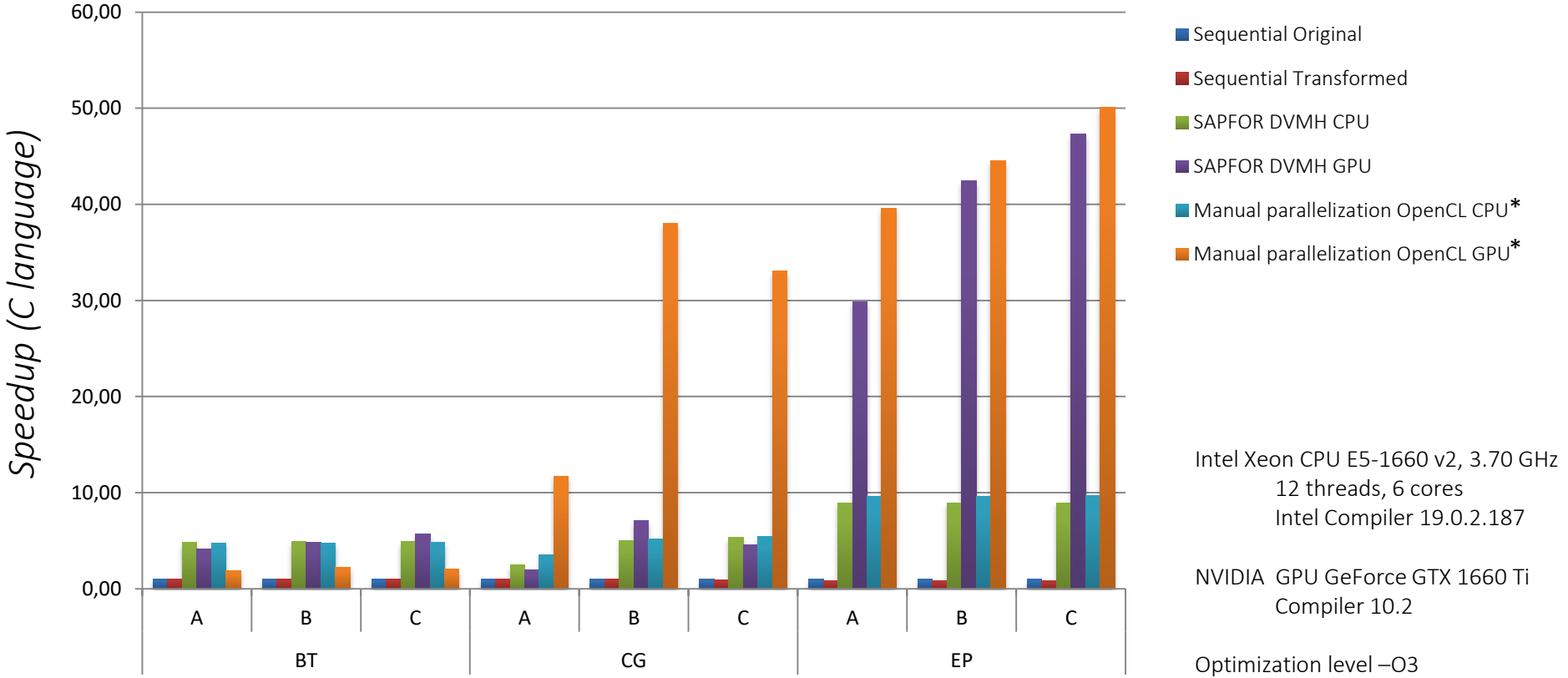
⇨

```
#pragma actual(B)
foo();
#pragma get_actual(A)

void foo() {…}
```

2.c

Parallelization of the NAS Parallel Benchmarks 3.3.1



Intel Xeon CPU E5-1660 v2, 3.70 GHz
12 threads, 6 cores
Intel Compiler 19.0.2.187

NVIDIA GPU GeForce GTX 1660 Ti
Compiler 10.2

Optimization level –O3

* Seo S., Jo G., Lee J. Performance Characterization of the NAS Parallel Benchmarks in OpenCL // 2011 IEEE International Symposium on. Workload Characterization (IISWC), 2011. — P. 137-148

Investigation of the NAS Parallel Benchmarks 3.3.1

Simple preliminary manual transformations to deal with the current limitations of SAPFOR (elimination of macros and merging of source files).

Manual elimination of time measurement functions outside the loop body.

Automatic source-level inline expansion:

- to ensure that functions do not capture pointer arguments (EP, BT),
- to ensure absence of data-dependencies (BT),
- to disambiguate pointer arguments (CG),
- to enable other transformations (EP).

Dynamic analysis to reveal privatizable arrays (BT, EP).

Manual specification of analysis options:

- to assume that subscript expression is in bounds value (BT),
- to prevent math functions to indicate errors by setting *errno* (EP).

Manual Transformation of the EP Benchmark

Replacement of a reduction array of the constant size with scalar variables:

```
q[l] = q[l] + 1.0;
```

⇨

```
switch (l) {
  case 0: q0 = q0 + 1.0; break;
  case 1: q1 = q1 + 1.0; break;
   ...
}
```

Elimination of a large privitizable array to utilize GPU:

```
for (i = 0; i < NK; i++) {
  ...
  x[i] = r46 * (*x4);
}

for (i = 0; i < NK; i++) {
  x1 = 2.0 * x[2 * i] - 1.0;
  x2 = 2.0 * x[2 * i + 1] - 1.0;
  ...
}
```

⇨

```
for (i = 0; i < NK; i++) {
  double x_2i, x_2i1;
  ...
  x_2i = r46 * (*x4);
  ...
  x_2i1 = r46 * (*x4);

  x1 = 2.0 * x_2i - 1.0;
  x2 = 2.0 * x_2i1 - 1.0;
  ...
}
```

Conclusion

SAPFOR implements an approach to the automation of parallel programming which follows an implicit parallel programming methodology and allows us to automatically produce DVMH versions of well-formed C programs.

To bring the program to a well-formed version SAPFOR provides source-to-source transformation techniques.

SAPFOR relies on the use of low-level program transformations, which are invisible to the programmer, to increase the quality of the analysis of the original program.

The proposed approach enables property sensitive transformations that means the internal representation of the program can be transformed into the most suitable form for program analysis.

The source code is available on GitHub: https://github.com/dvm-system

Thank you for attention

SCIENTIFIC CONFERENCE
Russian Supercomputing Days

http://dvm-system.org

https://github.com/dvm-system

KIAM RAS DVM SYSTEM

ОРДЕНА ЛЕНИНА ИМ.М.В.КЕЛДЫША РАН