



```
__global__ void kernelA(int ex, float* out)
{
    // Compute absolute (i,j) index
    // for the current GPU thread using
    int i = blockIdx.x * BLOCK_LENGTH;
    int j = blockIdx.y * BLOCK_LENGTH;

    // Compute one data point
    // for the kernel
    float val = 0.1f * 2.0f * ex;
    out[i * BLOCK_LENGTH + j] = val;
}
```

KernelGen генератор простых CUDA-ядер из кода на языке Fortran

Дмитрий Микушин

Причины создания

Необходимость портирования больших моделей на GPU:

- Очень много кода, требующего однообразных преобразований
- Множество версий одной и той же модели (портировать каждую отдельно?!)
- Многие разработчики моделей не готовы значительно менять код и поддерживать его (тем более – менять язык или завязываться на пропитеарные расширения)

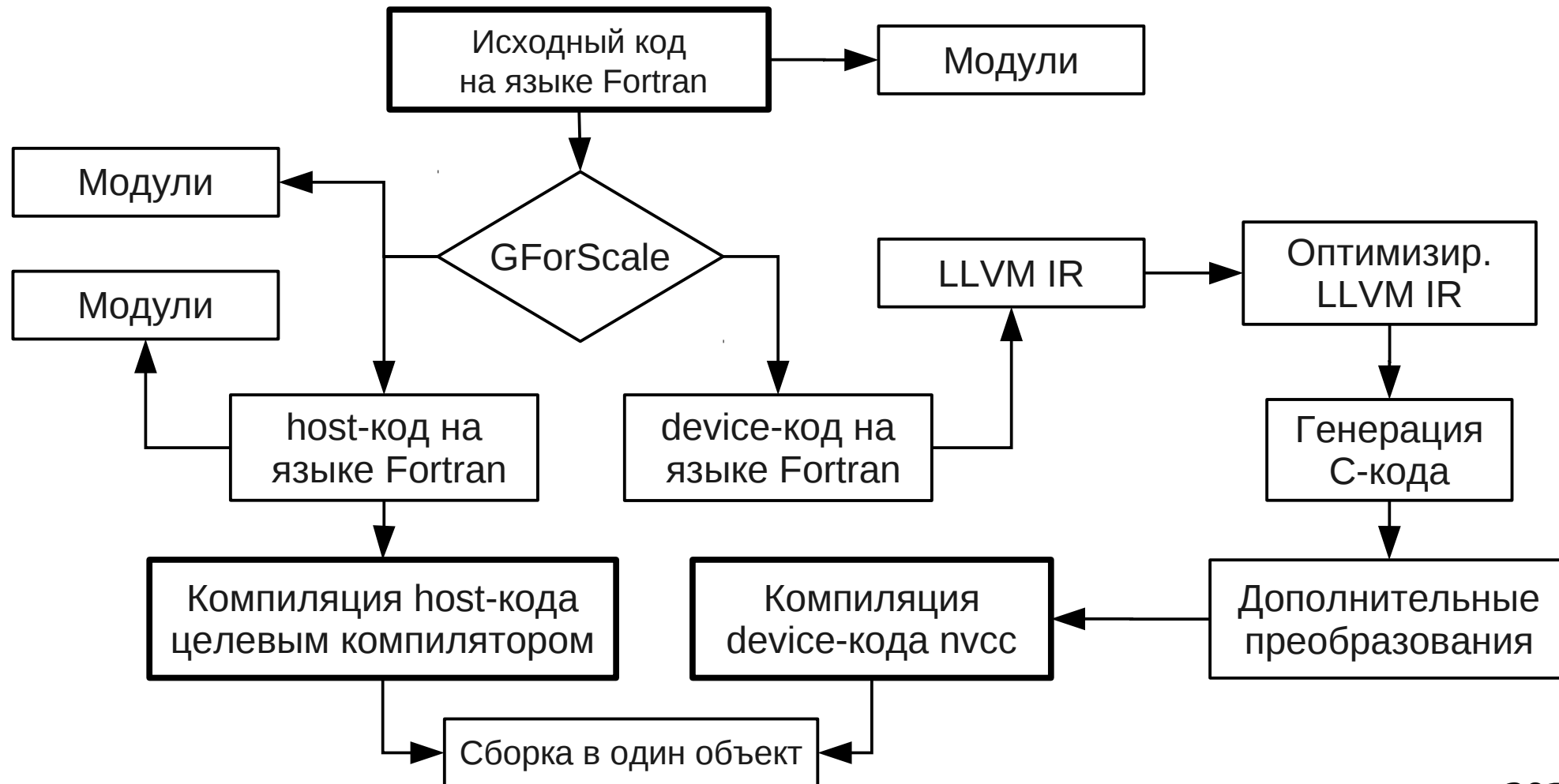
Задача

Портировать *параллельную* модель на GPU:

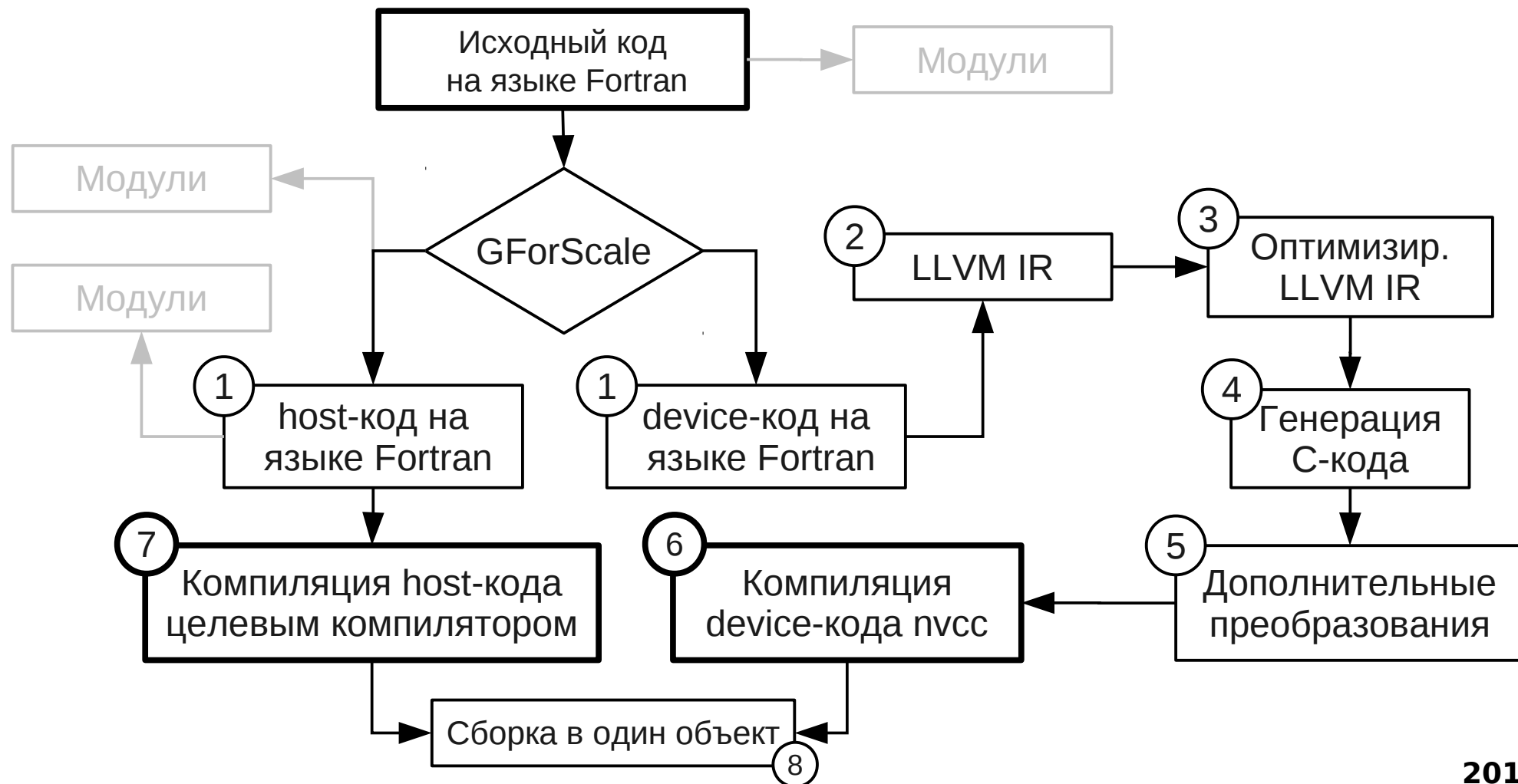
- С сохранением оригинального вида исходного кода
- Минимизировав количество ручной работы

Ключевое слово – “параллельную”, т.е. не нужно ничего заново распараллеливать, достаточно использовать уже имеющиеся алгоритмы на GPU.

Общая схема генератора



Всего - 8 основных шагов



Пример ахру

Рассмотрим действие генератора на простом примере:

```
subroutine axpy(n, a, x, y)

implicit none

integer, intent(in) :: n
real, intent(in) :: a, x(n)
real, intent(inout) :: y(n)

integer :: i

do i = 1, n
    y(i) = y(i) + a * x(i)
enddo

print *, 'Value of i after cycle = ', i

end subroutine axpy
```

Шаг 1

kgen-gforscale -Wk,--gforscale-scene-path=/opt/kgen/transforms/split/
-Wk,--gforscale-mode=tree axpy.f90

```
subroutine axpy_loop_1_gforscale(n, y, a, x)
implicit none
interface
subroutine axpy_loop_1_gforscale_blockidx_x(index, start, end)
bind(C)
use iso_c_binding
integer(c_int) :: index
integer(c_int), value :: start, end
end subroutine
subroutine axpy_loop_1_gforscale_blockidx_y(index, start, end)
bind(C)
use iso_c_binding
integer(c_int) :: index
integer(c_int), value :: start, end
end subroutine
end interface
integer :: i
integer, intent(in) :: n
real, intent(inout) :: y(n)
real, intent(in) :: a
real, intent(in) :: x(n)

#ifdef __CUDA_DEVICE_FUNC__
call axpy_loop_1_gforscale_blockidx_x(i, 1, n)
#else
do i = 1, n
#endif

    y(i) = y(i) + a * x(i)

#ifdef __CUDA_DEVICE_FUNC__
enddo
#endif

end subroutine axpy_loop_1_gforscale
```

device-код

```
subroutine axpy(n, a, x, y)
USE GFORSSCALE

implicit none

integer, intent(in) :: n
real, intent(in) :: a, x(n)
real, intent(inout) :: y(n)

integer :: i

!$GFORSSCALE SELECT axpy_loop_1_gforscale
if (gforscale_select(0, 1, 'axpy' // char(0))) then
!$GFORSSCALE CALL axpy_loop_1_gforscale
#ifdef __CUDA_DEVICE_FUNC__
    call gforscale_launch('axpy_loop_1_gforscale_' // char(0), &
        1, n, 0, 0, 4, n, sizeof(n), y, sizeof(y), a, sizeof(a), x, sizeof(x))
i = n + 1
#else
    call axpy_loop_1_gforscale(n, y, a, x)
#endif
!$GFORSSCALE END CALL axpy_loop_1_gforscale
else
!$GFORSSCALE LOOP axpy_loop_1_gforscale
do i = 1, n
    y(i) = y(i) + a * x(i)
enddo
!$GFORSSCALE END LOOP axpy_loop_1_gforscale
endif
!$GFORSSCALE END SELECT axpy_loop_1_gforscale

print *, 'Value of i after cycle = ', i

end subroutine axpy
```

host-код

2011

Шаг 2

```
dragonegg-gfortran -c axpy.axpy_loop_1_gforscale.F90  
-D__CUDA_DEVICE_FUNC__ -ffree-line-length-none  
-fplugin=/opt/llvm/dragonegg/lib64/dragonegg.so -O0 -S  
-fplugin-arg-dragonegg-emit-ir -o axpy.axpy_loop_1_gforscale.F90.bc
```

```
; ModuleID = 'axpy.axpy_loop_1_gforscale.F90'  
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-  
i64:64:64-f32:32:32-f64:64:64-v64:64:64-v128:128:128-a0:0:64-s0:64:64-  
f80:128:128-f128:128:128-n8:16:32:64"  
target triple = "x86_64-unknown-linux-gnu"
```

```
module asm "\09.ident\09\22GCC: (GNU) 4.5.4 20110527 (prerelease) LLVM:  
131968\22"
```

```
%"integer(kind=4)" = type i32  
%"real(kind=4)" = type float
```

```
define void @axpy_loop_1_gforscale_(i32* %n, [0 x float]* %y, float* %a, [0  
x float]* %x) nounwind {
```

```
entry:
```

```
  %n_addr = alloca i32*, align 8  
  %y_addr = alloca [0 x float]*, align 8  
  %a_addr = alloca float*, align 8  
  %x_addr = alloca [0 x float]*, align 8  
  %memtmp = alloca i32  
  %"alloca point" = bitcast i32 0 to i32  
  store i32* %n, i32** %n_addr  
  store [0 x float]* %y, [0 x float]** %y_addr  
  store float* %a, float** %a_addr  
  store [0 x float]* %x, [0 x float]** %x_addr  
  %0 = load i32** %n_addr, align 64  
  %1 = load [0 x float]** %y_addr, align 64  
  %2 = load float** %a_addr, align 64  
  %3 = load [0 x float]** %x_addr, align 64  
  %"ssa point" = bitcast i32 0 to i32  
  br label %"2"
```

```
"2":
```

```
  %4 = load i32* %0, align 4  
  %5 = sext i32 %4 to i64  
  %6 = icmp sge i64 %5, 0  
  %7 = select i1 %6, i64 %5, i64 0  
  %8 = add nsw i64 %7, -1  
  %9 = mul i64 %7, 32
```

```
; preds = %entry
```

```
  %10 = mul i64 %7, 4  
  %11 = load i32* %0, align 4  
  %12 = sext i32 %11 to i64  
  %13 = icmp sge i64 %12, 0  
  %14 = select i1 %13, i64 %12, i64 0  
  %15 = add nsw i64 %14, -1  
  %16 = mul i64 %14, 32  
  %17 = mul i64 %14, 4  
  %18 = load i32* %0, align 4  
  call void (i32*, i32, i32, ...) @axpy_loop_1_gforscale_blockidx_x(i32*  
noalias %memtmp, i32 1, i32 %18) nounwind  
  %19 = load i32* %memtmp, align 4  
  %20 = sext i32 %19 to i64  
  %21 = add nsw i64 %20, -1  
  %22 = load i32* %memtmp, align 4  
  %23 = sext i32 %22 to i64  
  %24 = add nsw i64 %23, -1  
  %25 = bitcast [0 x float]* %1 to float*  
  %26 = getelementptr float* %25, i64 %24  
  %27 = load float* %26, align 4  
  %28 = load float* %2, align 4  
  %29 = load i32* %memtmp, align 4  
  %30 = sext i32 %29 to i64  
  %31 = add nsw i64 %30, -1  
  %32 = bitcast [0 x float]* %3 to float*  
  %33 = getelementptr float* %32, i64 %31  
  %34 = load float* %33, align 4  
  %35 = fmul float %28, %34  
  %36 = fadd float %27, %35  
  %37 = bitcast [0 x float]* %1 to float*  
  %38 = getelementptr float* %37, i64 %21  
  store float %36, float* %38, align 4  
  br label %return
```

```
return:
```

```
  ret void  
}
```

```
; preds = %"2"
```

```
declare void @axpy_loop_1_gforscale_blockidx_x(i32* noalias, i32, i32, ...)
```


Шаг 3

```
/opt/llvm/bin/opt -std-compile-opts axpy.axpy_loop_1_gforscale.F90.bc  
-S -o axpy.axpy_loop_1_gforscale.F90.bc.opt
```

```
; ModuleID = 'axpy.axpy_loop_1_gforscale.F90.bc'  
target datalayout = "e-p:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-i64:64:64-f32:32:32-f64:64:64-v64:64:64-v128:128:128-a0:0:64-s0:64:64-f80:128:128-f128:128:128-n8:16:32:64"  
target triple = "x86_64-unknown-linux-gnu"  
  
module asm "\09.ident\09\22GCC: (GNU) 4.5.4 20110527 (prerelease) LLVM: 131968\22"  
  
define void @axpy_loop_1_gforscale_(i32* nocapture %n, [0 x float]* nocapture %y, float* %a, [0 x float]* %x) nounwind {  
entry:  
  %memtmp = alloca i32, align 4  
  %0 = load i32* %n, align 4  
  call void (i32*, i32, i32, ...)* @axpy_loop_1_gforscale_blockidx_x(i32* noalias %memtmp, i32 1, i32 %0) nounwind  
  %1 = load i32* %memtmp, align 4  
  %2 = sext i32 %1 to i64  
  %3 = add nsw i64 %2, -1  
  %4 = getelementptr [0 x float]* %y, i64 0, i64 %3  
  %5 = load float* %4, align 4  
  %6 = load float* %a, align 4  
  %7 = getelementptr [0 x float]* %x, i64 0, i64 %3  
  %8 = load float* %7, align 4  
  %9 = fmul float %6, %8  
  %10 = fadd float %5, %9  
  store float %10, float* %4, align 4  
  ret void  
}  
  
declare void @axpy_loop_1_gforscale_blockidx_x(i32* noalias, i32, i32, ...)
```

Шаг 4

```
/opt/llvm/bin/llc -march=c axpy.axpy_loop_1_gforscale.F90.bc.opt  
-o axpy.axpy_loop_1_gforscale.F90.bc.cu
```

```
asm("\t.ident\t\"GCC: (GNU) 4.5.4 20110527 (prerelease) LLVM: 131968\"\n"  
"" );  
  
...  
  
#ifdef __CUDA_DEVICE_FUNC__  
__device__  
#endif  
void axpy_loop_1_gforscale_(unsigned int *llvm_cbe_n, struct l_unnamed0 (*llvm_cbe_y), float *llvm_cbe_a, struct l_unnamed0 (*llvm_cbe_x));  
#ifdef __CUDA_DEVICE_FUNC__  
__device__  
#endif  
void axpy_loop_1_gforscale_blockidx_x(unsigned int *, unsigned int , unsigned int );  
  
void axpy_loop_1_gforscale_(unsigned int *llvm_cbe_n, struct l_unnamed0 (*llvm_cbe_y), float *llvm_cbe_a, struct l_unnamed0 (*llvm_cbe_x)) {  
    unsigned int llvm_cbe_memtmp; /* Address-exposed local */  
    unsigned int llvm_cbe_tmp_1;  
    unsigned int llvm_cbe_tmp_2;  
    unsigned long long llvm_cbe_tmp_3;  
    float *llvm_cbe_tmp_4;  
    float llvm_cbe_tmp_5;  
    float llvm_cbe_tmp_6;  
    float llvm_cbe_tmp_7;  
  
    llvm_cbe_tmp_1 = *llvm_cbe_n;  
    axpy_loop_1_gforscale_blockidx_x((&llvm_cbe_memtmp), lu, llvm_cbe_tmp_1);  
    llvm_cbe_tmp_2 = *(&llvm_cbe_memtmp);  
    llvm_cbe_tmp_3 = (((unsigned long long )(((unsigned long long )(((signed long long )(signed int )llvm_cbe_tmp_2))) + ((unsigned long long )  
18446744073709551615ull))));  
    llvm_cbe_tmp_4 = (&(*llvm_cbe_y).array[(((signed long long )llvm_cbe_tmp_3))];  
    llvm_cbe_tmp_5 = *llvm_cbe_tmp_4;  
    llvm_cbe_tmp_6 = *llvm_cbe_a;  
    llvm_cbe_tmp_7 = *((&(*llvm_cbe_x).array[(((signed long long )llvm_cbe_tmp_3))]);  
    *llvm_cbe_tmp_4 = (((float )(llvm_cbe_tmp_5 + (((float )(llvm_cbe_tmp_6 * llvm_cbe_tmp_7))))));  
    return;  
}
```

Шаг 5

```
asm( "\t.ident\t\"GCC: (GNU) 4.5.4 20110527 (prerelease) LLVM: 131968 \\\"\\n\"  
\"");  
  
...  
  
#ifdef __CUDA_DEVICE_FUNC__  
extern "C" __global__  
#endif  
void axpy_loop_1_gforscale_(unsigned int *llvm_cbe_n, struct l_unnamed0 (*llvm_cbe_y), float *llvm_cbe_a, struct l_unnamed0 (*llvm_cbe_x));  
#ifdef __CUDA_DEVICE_FUNC__  
__device__  
#endif  
void axpy_loop_1_gforscale_blockidx_x( unsigned int* index, unsigned int start, unsigned int end) { *index = blockIdx.x + start; }  
  
void axpy_loop_1_gforscale_(unsigned int *llvm_cbe_n, struct l_unnamed0 (*llvm_cbe_y), float *llvm_cbe_a, struct l_unnamed0 (*llvm_cbe_x)) {  
    unsigned int llvm_cbe_memtmp; /* Address-exposed local */  
    unsigned int llvm_cbe_tmp_1;  
    unsigned int llvm_cbe_tmp_2;  
    unsigned long long llvm_cbe_tmp_3;  
    float *llvm_cbe_tmp_4;  
    float llvm_cbe_tmp_5;  
    float llvm_cbe_tmp_6;  
    float llvm_cbe_tmp_7;  
  
    llvm_cbe_tmp_1 = *llvm_cbe_n;  
    axpy_loop_1_gforscale_blockidx_x(&llvm_cbe_memtmp, 1u, llvm_cbe_tmp_1);  
    llvm_cbe_tmp_2 = *(&llvm_cbe_memtmp);  
    llvm_cbe_tmp_3 = (((unsigned long long )(((unsigned long long )(((signed long long )(signed int )llvm_cbe_tmp_2))) + ((unsigned long long )  
18446744073709551615ull ))));  
    llvm_cbe_tmp_4 = (&(*llvm_cbe_y).array[((( signed long long )llvm_cbe_tmp_3))];  
    llvm_cbe_tmp_5 = *llvm_cbe_tmp_4;  
    llvm_cbe_tmp_6 = *llvm_cbe_a;  
    llvm_cbe_tmp_7 = *(&(*llvm_cbe_x).array[((( signed long long )llvm_cbe_tmp_3))]);  
    *llvm_cbe_tmp_4 = (((float )(llvm_cbe_tmp_5 + (((float )(llvm_cbe_tmp_6 * llvm_cbe_tmp_7))))));  
    return;  
}
```

Шаги 6-8

Компиляция отдельных частей и компоновка в общий объектный файл:

6

```
nvcc -g -c axpy.axpy_loop_1_gforscale.F90.bc.cu -D__CUDA_DEVICE_FUNC__ -G -o  
axpy.axpy_loop_1_gforscale.F90.o
```

7

```
gfortran -g -c axpy.host.F90 -D__CUDA_DEVICE_FUNC__ -ffree-line-length-none  
-I/opt/kgem/include -o axpy.host.F90.o
```

8

```
/usr/bin/ld --unresolved-symbols=ignore-all -r -o axpy.o_kgem axpy.host.F90.o  
axpy.axpy_loop_1_gforscale.F90.o
```

Тестирование ахру

По умолчанию – исполнение на CPU

```
[marcusmae@T61p ахру]$ ./ахру
Usage: ./ахру <n> <eps>
[marcusmae@T61p ахру]$ ./ахру 10 0.001
Value of i after cycle =          11
Max abs diff = 0.000000
```

**# Использование GPU при включении переменной окружения
(доп. отладочная печать отображения аргументов в память GPU)**

```
[marcusmae@T61p ахру]$ ./ахру 1000 0.001
arg 0 maps memory segment [140735344500736 .. 140735344508928] to
[1052672 .. 1060864]
arg 1 maps memory segment [28172288 .. 28184576] to [1060864 .. 1073152]
arg 2 reuses mapping created by arg 0
arg 3 reuses mapping created by arg 1
Value of i after cycle =          1001
Max abs diff = 0.000000
```


Структура системы

/opt/kgen/

bin/

g95xml-refids

g95xml-tree

kgen

kgen-gforscale

kgen-gfortran



XML-разметка кода

Скрипт, вып. шаги 1-8

Преобразование кода

Фронтенд (“компилятор”)

include/

gforscale.h

gforscale.mod

gforscale.dragonegg.mod



Runtime-модули

lib/

libgforscale.so



Runtime-библиотека

transforms/

split/



Правила преобразования
кода

Хочу попробовать!

Генератор доступен на серверах СибНИГМИ и CUDA.CS.MSU.SU

Загрузка исходного кода с примерами

```
[dmikushin@tesla-cmc ~]$ svn co svn+ssh://dmikushin@tesla.parallel.ru/svn/kernelgen
dmikushin@tesla.parallel.ru's password:
dmikushin@tesla.parallel.ru's password:
...

[dmikushin@tesla-cmc ~]$ cd kernelgen/tests/public/logexp/
[dmikushin@tesla-cmc logexp]$ make
...

[dmikushin@tesla-cmc logexp]$ ./logexp 1000 1000 0.001
arg 0 maps memory segment [140734329888768 .. 140734329896960] to [1052672 .. 1060864]
arg 1 reuses mapping created by arg 0
arg 2 maps memory segment [139978601713664 .. 139978613723136] to [1060864 .. 13070336]
arg 3 reuses mapping created by arg 2
arg 4 reuses mapping created by arg 2
  Value of i after cycle =      1001
  Value of j after cycle =      1001
Max abs diff = 0.000002
```

Что дальше?

Применение генератора к модели COSMO:

1. Компиляция с помощью kgen-gfortran, исправление приоритетных ошибок
2. Последовательное подключение device-реализаций параллельных циклов, сравнение результатов (контроль багов в C backend)
3. Доработка системы для оптимизации и поддержки multi-GPU

На 1-2 – ближайшие 3 недели