

# Диалоговый генератор CUDA-кода

Языки программирования и компиляторы '2017  
Всероссийская научная конференция памяти А.Л. Фуксмана

**РЛС-2017**

Аллазов А.Н. Гуда С.А. Морылев Р.И.  
Институт математики, механики и компьютерных наук  
Южный федеральный университет



# Диалоговый высокоуровневый оптимизирующий распараллеливатель программ (ДВОР)

The screenshot displays the IHOP (Interactive High-Level Parallel Optimizer) interface. The main window shows a C code snippet for calculating correlation. A context menu is open over a loop, listing various optimization techniques. A table at the bottom shows the results of these optimizations, including a GPGPU parallelization suggestion.

```
kernel_correlation
}
mean[j] = (mean[j] / float_n)
}
for j = 0; j < m; j = (j + 1)
{
stddev[j] = 0.
for i = 0; i < n; i = (i + 1)
{
stddev[j] = (stddev[j] + (a[i, j] - mean[j]))
}
stddev[j] = (stddev[j] / float_n)
stddev[j] = sqrt(stddev[j])
if stddev[j] <= eps
{
__uni49 = 1.
}
else
{
__uni49 = stddev[j]
}
stddev[j] = __uni49
}
for i = 0; i < n; i = (i + 1)
{
```

Context Menu:

- Open Selector
- XML Dump
- Build DepGraph
- Build CallGraph
- Arithmetic Operator Expansion
- Exception Generator
- Full inline substitution
- Subroutine splitting
- Corruptor
- Loop Distribution
- Loop Fragmentation
- Loop Full Unrolling
- Loop Header Removal
- Loop Nesting
- Loop Unrolling
- Recurrency Elimination
- Strip Mining
- Execute on GPU
- Parallel execute on GPU

Category	Description
Loops	Loop iterations can be executed independently
Loops	Loop iterations can be executed independently
Loops	Loop is sequential
Loops	Loop is sequential
Loops	Loop is sequential
Loops	Loop iterations can be executed independently
Loops	Loop is sequential
Loops	Loop iterations can be executed independently
Loops	Loop iterations can be executed independently
Loops	Loop iterations can be executed independently
Loops	Loop iterations can be executed independently
GPGPU	The loop can be executed in parallel. Do you want to parallelize it using GPGPU computations?
Loops	Loop is sequential
Loops	Loop iterations can be executed independently
Loops	Loop iterations can be executed independently
Loops	Loop iterations can be executed independently

Parser Output | Issues | GPGPU

# Этапы работы генератора GPU-кода

- 1) инициализация параметров участков кода, предварительные проверки и преобразования;
- 2) анализ участков кода и определение параметров отображения кода на ускоритель;
- 3) GPU-преобразования кода;
- 4) получение текстового представления кода.

# Результат работы генератора

```
for (int i0 = 0; i0 < n0; i0++)  
  for (int i1 = 0; i1 < n1(i0); i1++)  
    for (int i2 = 0; i2 < n2(i0, i1); i2++)  
      LoopBodyBlock
```



```
if (n0 > 0 && n_max1 > 0 && n_max2 > 0)  
{  
  dim3 blockDim = dim3(b0, b1, b2);  
  dim3 gridDim = dim3(g0, g1, g2);  
  kernel1<<<gridDim, blockDim>>>(...);  
  gpuErrchk(cudaPeekAtLastError());  
}  
  
__global__ void kernel1(...)  
{  
  iψ(0) = blockIdx.x*blockDim.x+threadIdx.x  
  iψ(1) = blockIdx.y*blockDim.y+threadIdx.y  
  iψ(2) = blockIdx.z*blockDim.z+threadIdx.z  
  if (i0<n0 && i1<n1(i0) && i2<n2(i0, i1))  
    LoopBodyBlock  
}
```

# Анализ данных

Для каждого фрагмента кода общие для CPU и GPU ячейки памяти мы делим на три непересекающихся типа:

- внешние (для фрагмента) скалярные переменные, значения которых не меняются;
- внешние изменяемые скалярные переменные;
- массивы.

# Расстановка синхронизаций общих ячеек памяти

## Пример: LU-разложение матрицы

```
initMatrix(A); //initialization
for (int k = 0; k < N; k++)
{
    //fragment 1 begin
    for (int j = k + 1; j < N; j++)
        A[k*N+j] = A[k*N+j] / A[k*N+k];
    //fragment 1 end

    //fragment 2 begin
    for(int i = k + 1; i < N; i++)
        for (int j = k + 1; j < N; j++)
            A[i*N+j] = A[i*N+j] - A[i*N+k] * A[k*N+j];
    //fragment 2 end
}
```

# Расстановка синхронизаций общих ячеек памяти

- Начальное расположение cudaMemcpy: после CPU и GPU-фрагментов для измененных в этом фрагменте ячеек
- Алгоритм оптимизации синхронизаций данных сдвигает cudaMemcpy вниз по графу потока управления, выносит их из циклов и удаляет дублирующиеся
- Как не допустить ошибок?

# Расстановка синхронизаций общих ячеек памяти

Нужно соблюсти правило:

*Рассмотрим путь на графе потока управления, соединяющий оператор на  $dev1$ , изменяющий некоторую общую для  $dev1$  и  $dev2$  переменную  $x$ , и оператор на  $dev2$  с вхождением той же ячейки памяти.*

*На каждом таком пути должна располагаться хотя бы одна операция копирования переменной  $x$  в направлении  $dev1 \rightarrow dev2$ , срабатывающая раньше любой операции копирования в противоположном  $dev2 \rightarrow dev1$  направлении, если последняя присутствует на данном пути.*

# Пример: LU-разложение матрицы

```
gpuErrchk(cudaMalloc(&A_gpu, N*N*8));
gpuErrchk(cudaMemcpy(A_gpu, A, N*N*8, cudaMemcpyHostToDevice));
for (int k = 0; k < N; k = k + 1)
{
    if (N - (k + 1) > 0)
    {
        int blockDim = 256;
        int gridDim = (N-k-1+blockDim-1)/blockDim;
        kernel0<<<gridDim,blockDim>>>(k,A_gpu,n);
        gpuErrchk(cudaPeekAtLastError());
    }
    if (N - (k + 1) > 0 && N - (k + 1) > 0)
    {
        dim3 blockDim = dim3(64,4,1);
        dim3 gridDim = dim3(
            (N-k-1+blockDim.x-1)/blockDim.x,
            (N-k-1+blockDim.y-1)/blockDim.y, 1);
        kernel1<<<gridDim,blockDim>>>(k, A_gpu, N);
        gpuErrchk(cudaPeekAtLastError());
    }
}
gpuErrchk(cudaMemcpy(A, A_gpu, N*N*8, cudaMemcpyDeviceToHost));
```

# Отображение циклов на измерения пространства потоков

```
for (int i0 = 0; i0 < n0; i0++)  
  for (int i1 = 0; i1 < n1(i0); i1++)  
    for (int i2 = 0; i2 < n2(i0, i1); i2++)  
      LoopBodyBlock
```



```
i0 = blockIdx.x*blockDim.x+threadIdx.x  
i1 = blockIdx.y*blockDim.y+threadIdx.y  
i2 = blockIdx.z*blockDim.z+threadIdx.z
```

**Плюс еще 5 способов. Какой выбрать?**

# Статическая профилировка

- Определим коэффициенты при счетчиках циклов в каждом обращении к массиву  $X[a_0*i_0 + a_1*i_1 + a_2*i_2 + p]$
- Будем различать 3 типа коэффициентов:
  - $a=0$
  - $a=\text{const} < \text{Cache size}$
  - $a$  – unknown или  $\geq \text{Cache size}$
- Оценим время доступа к памяти для всех комбинаций типов коэффициентов

# Статическая профилировка

$$\begin{aligned}t_{11} &= 0, & t_{12} &= \min((CS/a_1)b_0, BV)^{-1}, & t_{13} &= b_0^{-1}, \\t_{21} &= \min((CS/a_0)b_1, BV)^{-1}, \\t_{22} &= \min(CS/(a_0b_0a_1)b_0 + CS\%(a_0b_0a_1)/a_0, BV)^{-1}, \\t_{23} &= \min(CS/a_0, b_0)^{-1}, & t_{31} &= b_1^{-1}, & t_{32} &= 1, & t_{33} &= 1.\end{aligned}$$

Время – в расчете на 1 поток

За единицу времени принято время чтения одной кеш-линейки.

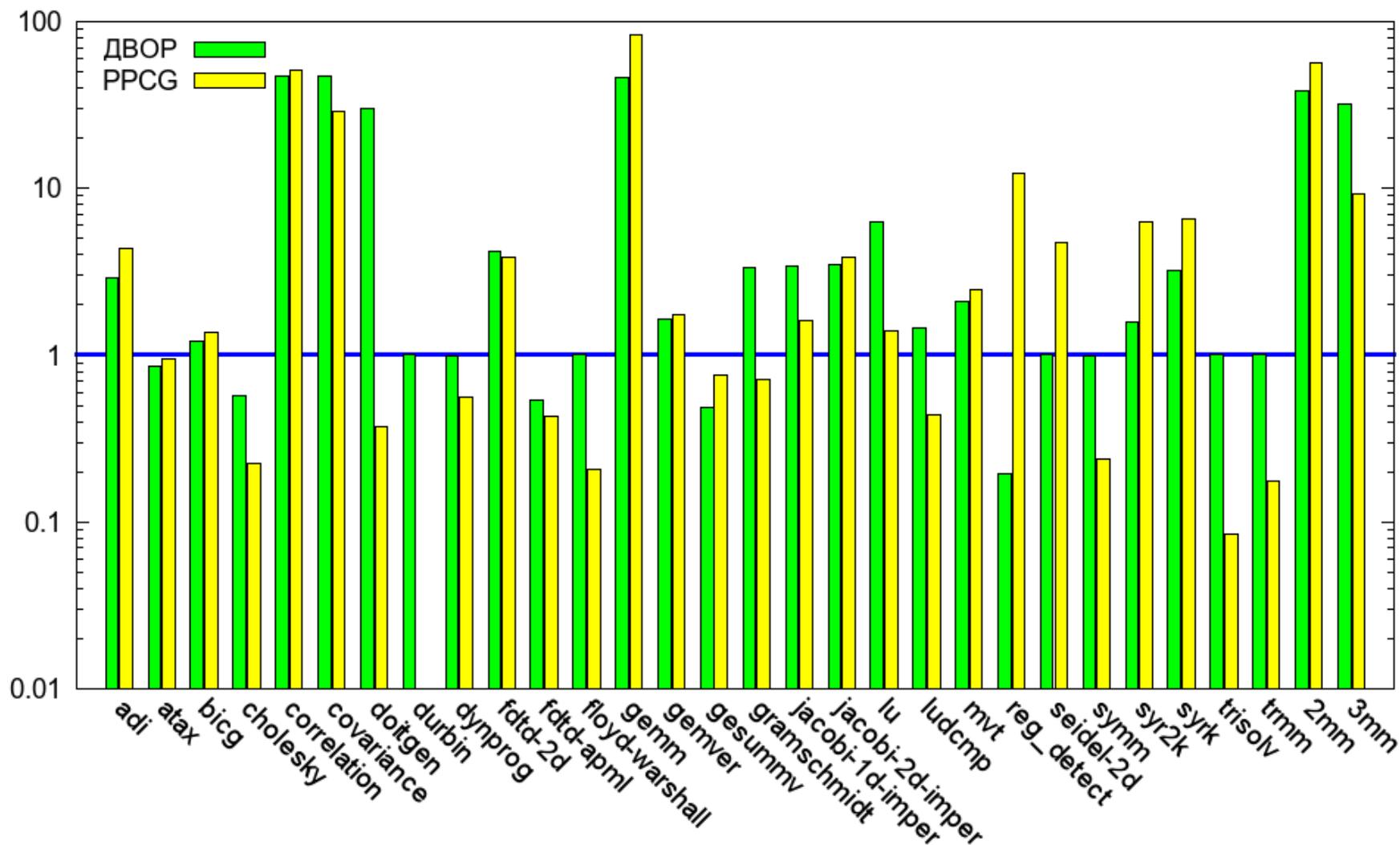
Отображение циклов на измерения пространства потоков - естественное

# Статическая профилировка

Для 3х-мерного гнезда:

$$\begin{aligned}t_{111} &= 0, & t_{121} &= \min((CS/a_1)b_0b_2, BV)^{-1}, \\t_{131} &= (b_0b_2)^{-1}, & t_{211} &= \min((CS/a_0)b_1b_2, BV)^{-1}, \\t_{221} &= \min((CS/(a_0b_0a_1)b_0 + CS\%(a_0b_0a_1)/a_0)b_2, b_0b_1)^{-1}, \\t_{231} &= \min((CS/a_0)b_2, b_0b_2)^{-1}, & t_{311} &= (b_1b_2)^{-1}, \\t_{321} &= b_2^{-1}, & t_{331} &= b_2^{-1}, & t_{112} &= \min((CS/a_2)b_0b_1, BV)^{-1}, \\t_{122} &= \min((CS/(a_1b_1a_2)b_1 + CS\%(a_1b_1a_2)/a_1)b_0, BV)^{-1}, \\t_{132} &= b_0^{-1}, \\t_{212} &= \min((CS/(a_0b_0a_2)b_0 + CS\%(a_0b_0a_2)/a_0)b_1, BV)^{-1}, \\t_{222} &= \min(CS/(a_0b_0a_1b_1a_2)b_0b_1 + \\&\quad + CS\%(a_0b_0a_1b_1a_2)/(a_0b_0a_1)b_0 + \\&\quad + CS\%(a_0a_1b_0b_1a_2)\%(a_0b_0a_1)/a_0), BV)^{-1}, \\t_{232} &= \min(CS/a_0, b_0)^{-1}, & t_{312} &= b_1^{-1}, & t_{322} &= 1, & t_{332} &= 1, \\t_{113} &= (b_0b_1)^{-1}, & t_{123} &= \min((CS/a_1)b_0, b_0b_1)^{-1}, \\t_{133} &= b_0^{-1}, & t_{213} &= \min((CS/a_0)b_1, b_0b_1)^{-1}, \\t_{223} &= \min(CS/(a_0b_0a_1)b_0 + CS\%(a_0b_0a_1)/a_0, b_0b_1)^{-1}, \\t_{233} &= \min(CS/a_0, b_0)^{-1}, & t_{313} &= b_1^{-1}, & t_{323} &= 1, & t_{333} &= 1.\end{aligned}$$

# Сравнение с PPCG на Polybench



Ускорение сгенерированного кода для GPU Tesla C2075 по сравнению с CPU Intel Core i7-3820 3.60GHz (компилятор GCC с флагом -O2)

# <http://ops.opsgroup.ru/>

## АВТОМАТИЧЕСКИЙ РАСПАРАЛЛЕЛИВАТЕЛЬ

ОРС ВЕБ ИНТЕРФЕЙС

ВЕБ-РАСПАРАЛЛЕЛИВАТЕЛЬ

ДОКУМЕНТАЦИЯ

Язык: Русский / [English](#)

### Web-распараллеливатель

Предыдущий шаг

Следующий шаг

#### Шаг 1

Режим преобразования

#### Шаг 2

Источник программы

#### Шаг 3

Ввод программы

#### Шаг 4

Выполнение преобразования

### Выберите режим преобразования

- Автоматическая генерация MPI-кода с размещением данных [?]
- Автоматическое распределение данных под кэш-память [?]
- Автоматическая генерация OpenMP-кода [?]
- Генератор GPU-кода (CUDA)

# Директива #pragma ops target

```
/* E := A*B */
#pragma ops target collapse(2)
for (i = 0; i < _PB_NI; i++)
  for (j = 0; j < _PB_NJ; j++)
  {
    E[i][j] = 0;
    for (k = 0; k < _PB_NK; ++k)
      E[i][j] += A[i][k] * B[k][j];
  }

/* F := C*D */
#pragma ops target collapse(2)
for (i = 0; i < _PB_NJ; i++)
  for (j = 0; j < _PB_NL; j++)
  {
    F[i][j] = 0;
    for (k = 0; k < _PB_NM; ++k)
      F[i][j] += C[i][k] * D[k][j];
  }

/* G := E*F */
#pragma ops target collapse(2)
for (i = 0; i < _PB_NI; i++)
  for (j = 0; j < _PB_NL; j++)
  {
    G[i][j] = 0;
    for (k = 0; k < _PB_NJ; ++k)
      G[i][j] += E[i][k] * F[k][j];
  }
```