Ускорение операции быстрой свёртки для множества комплексных векторов на основе технологии OpenCL

А.А. Бурцев

НИЦ «Курчатовский институт» — НИИСИ, Москва, Российская Федерация;

burtsev@niisi.msk.ru

Аннотация. Статья посвящена применению технологии OpenCL, позволяющей использовать мощные ресурсы графических процессоров для повышения быстродействия вычислительных программ. Рассматриваются приёмы разработки в среде OpenCL эффективных параллельных программ, позволяющих ускорить операцию свёртки для множества комплексных векторов на основе быстрого преобразования Фурье.

Ключевые слова: параллельное программирование, технология OpenCL, гетерогенные системы, быстрое преобразование Фурье, операция свёртки.

1. Введение

На современных компьютерных установках для разработки параллельных программ обычно применяются широко известные технологии, ориентированные на исполнение таких программ в многопроцессорной среде (OpenMP [1, п. 5.1]) или в среде из нескольких связанных сетью компьютеров (MPI [1, п. 5.2]). Но наряду с ними предлагаются также и такие технологии, которые позволяют использовать мощные ресурсы графических процессоров для повышения быстродействия вычислительных программ. Так что сегодня можно ускорять вычисления даже на одном компьютере, если в его составе предусмотрена видеокарта, поддерживающая одну из этих технологий. Например, технологию OpenCL [2].

Знакомство с технологией OpenCL было предложено автором в серии ранее опубликованных им статей [3-7]. В них демонстрировались приёмы разработки эффективных параллельных программ для решения ряда вычислительных задач. В частности, рассматривались варианты построения по технологии OpenCL программ, ускоряющих перемножение больших матриц [3], вычисление интегралов [4], а также программ, ускоряющих в среде OpenCL выполнение операции быстрого преобразования Фурье (БПФ) как для одиночных комплексных векторов [5-6], так и для их множества [7].

Причём для множества векторов получалось добиться ускорения операции БПФ даже в большей степени, чем для одиночных векторов, если удавалось подключить к обработке вычислительной программы в среде OpenCL как можно больше параллельных исполнителей.

Решение многих задач цифровой обработки сигналов строится на основе применения операции быстрого преобразования Фурье. И если уж с помощью технологии OpenCL можно значительно ускорить выполнение самой операции БПФ, то следующим встаёт вопрос, а в какой степени удастся ускорить те вычислительные программы, которые используют БПФ в качестве базовой операции?

Проведём исследование этого вопроса на примере разработки одной из таких программ. Рассмотрим программу, выполняющую операцию быстрой свёртки, выполняемой над множеством комплексных векторов. В такой программе для выполнения свёртки каждой пары комплексных векторов требуется (помимо других операций) по три раза вызывать операцию БПФ для векторов той же длины.

Как и прежде, представим сначала алгоритмы, по которым осуществляется такая множественная операция быстрой свёртки в обычных программах, рассчитанных на последовательное их исполнение одним процессором. А затем рассмотрим разнообразные варианты программ, позволяющих ускорить исполнение такой операции множественной свёртки в среде OpenCL. А чтобы проанализировать показатели производительности OpenCL-программ разных вариантов, прогоним варианты этих программ на различных OpenCL-платформах для одних и тех же наборов данных, построим для наглядности таблицы полученных коэффициентов ускорения, и затем, сравнив их результаты, выясним, какой из вариантов OpenCL-программы обеспечивает наилучшие показатели ускорения на той или иной OpenCL-платформе.

2. Базовая программа быстрой свёртки комплексных векторов многомерного массива

Операция свёртки двух комплексных векторов X[L] длины L и Y[S] длины S заключается в получении третьего вектора Z[N] длиной N=L+S-1, элементы Z_K (k=0,..,N-1) которого вычисляются по формуле [8,с.147]:

$$Z_k = \sum_{i=0}^{S-1} \{Y_i \times X_{k-i}\}$$

Операция свёртки (convolution) над векторами (последовательностями чисел) обозначается сокращённо символом * Z[N]=X[L]*Y[S].

Сложность вычисления свёртки по этой формуле оценивается порядком O(N·S). Можно, однако, заметно ускорить вычисление свёртки путём применения быстрого преобразования Фурье (БПФ). Такой вариант вычисления получил название быстрая свёртка [8,с.355]. Поясним подробнее, как она осуществляется.

Заданные векторы X[L] и Y[S] дополним нулями до длины ближайшей степени двойки ($N=2^P$, N≥L+S-1). Применим к каждому вектору X[N] и Y[N] операцию БПФ (*FFT*):

X'[N]= *FFT*(X[N]); Y'[N]= *FFT*(Y[N]);

Получившиеся векторы X'[N] и Y'[N] перемножим покомпонентно, получив вектор Z'[N]:

$Z'_K = X'_K \times Y'_K$ для всех k=0,..,N-1.

А теперь применим к вектору Z' обратное БПФ (*iFFT*) и получим вектор Z[N]:

Z[N]= *iFFT*(Z'[N]),

который и будет результатом свёртки.

Сложность вычисления быстрой свёртки по такому алгоритму оценивается порядком $O(N \cdot \log_2 N)$, как и сама операция БПФ.

С алгоритмом прямого БПФ мы уже не раз знакомились ранее (например, см. описание функции FFT в [5, п.2.2]). Он основан на многократном исполнении так называемых операций «бабочек Фурье» BF(A,B,V) над всевозможными парами комплексных чисел A и B с весовым коэффициентом V вида:

$$W_N^u = e^{-i \cdot 2\pi \cdot u/N}$$

Для обратного БПФ нужно исполнить такой же алгоритм БПФ, но в качестве весовых коэффициентов взять сопряженные им комплексные числа (т.е. с другим знаком в мнимой части):

$$W_N^u = e^{+i \cdot 2\pi \cdot u/N}$$

а в завершении провести так называемую нормализацию, поделив значения полученных элементов вектора на величину его длины N.

Предполагается, что значения весовых коэффициентов, используемые алгоритмом БПФ, вычисляются заранее и приготавливаются в виде таблицы – массива комплексных чисел W (размером от 0 до N). Причём значение, сопряжённое значению W[u], можно взять из этой таблицы на такой же позиции с конца: W[N-u].

Учитывая это, подкорректируем алгоритм функции FFT. Добавим в заголовок функции параметр D, задающий направление БПФ: D=1 для прямого и D=0 для обратного преобразования Фурье. И будем выбирать коэффициент V из таблицы W, проверяя условие: если D=1, то V=W[u], иначе V=W[N-u]. А при завершении всего преобразования добавим выполнение нормализации, если D=0. После такой модификации функцию FFT можно будет применять для выполнения не только прямого (с D=1), но и обратного (с D=0) БПФ.

```
void FFT(int N,complex *X,
 complex *Y,complex *W, int D){
complex *VX,*VY,V,T;
int P= iLog2(N); float Norm;
//1. Бит-реверсная перестановка:
 BRVPRST(N,P,X,Y);
//2.Основной цикл исполнения бабочек Фурье:
 int t,s,h,G,R,d,k,u,a,b;
 s=1;h=2;G=1;R=N/2;d=N/2;
 for (t=1; t<=P; t++) {
 // s=2^(t-1); h=2^t; G=2^(t-1); R=2^(P-t)
 for(k=0,u=0;k<G;k++,u=u+d){
 if(D) V=W[u];else V=W[N-u];
 for(i=0,a=k; i<R; i++,a=a+h)
 {b=a+s; BF(Y[a],Y[b],V);}
 }//for k
 h=h*2;s=s*2;G=G*2;R=R/2;d=d/2;
 }//for t
// 3. Нормализация (для обратного БПФ)
 if(!D){ Norm=1.00/(float)N;
 for (i=0;i<N;i++)
 Y[i]= cMScal(Y[i],Norm);
 }//if !Dir
}//FFT
```

В теле этой функции используется вызов макроопределения **BF** для исполнения операции «бабочки Фурье» и функция **cMScal** для умножения комплексного числа на вещественный скаляр. Приведём здесь их возможные реализации (для версии языка Си, имеющей встроенный тип complex):

```
#define BF(A,B,V) { complex T; \
 T=B*V; B=A-T; A=A+T; }
complex cMScal(complex C, float S)
{ complex R; R.Re= C.Re*S;
 R.Im= C.Im*S; return R; }
```

Для бит-реверсного копирования комплексного вектора применяется вспомогательная процедура **BRVPRST**:

```
void BRVPRST(int N, int L,
  complex *X,complex *Y) {
 unsigned int i,j;
 for (i=0; i<N; i++) {
 BRvrsVal(j,i,L); Y[j]=X[i];
 }// for i
}//BRVPRST
```

Макрос **BRvrsVal**(j,i,L) здесь используется для получения бит-реверсного значения j длины L (т.е. записанного в обратном порядке значения в битовом коде длины L) для заданного индекса i. Приведём один из возможных вариантов определения такого макроса:

```
#define BRvrsVal(k,i,L){ k=i;\
k=(k&0x55555555)<<1|(k&0xAAAAAAAA)>>1;\
k=(k&0x33333333)<<2|(k&0xCCCCCCCC)>>2;\
k=(k&0x0F0F0F0F)<<4|(k&0xF0F0F0F0)>>4;\
k=(k&0x00FF00FF)<<8|(k&0xFF00FF00)>>8;\
k=(k&0x0000FFFF)<<16|(k& 0xFFFF0000)>>16;\
k>>=(32-L); }
```

Для исполнения БПФ множества комплексных векторов длиной N, уложенных в матрицу размером M×J, преобразуем функцию MFFT, представленную ранее в [7]. Для этого дополним заголовок функции параметром D, а внутрь двух вложенных циклов for (по m и j) поместим вызов модифицированной функции FFT:

```
void MFFT(int M,int J,int N,int D,
complex *X,complex *Y,complex *W){
int m,j,mj;
 for (m=0; m<M; m++)
 for (j=0;j<J;j++){ mj=(m*J+j)*N;
 FFT(N,&X[mj],&Y[mj],W,D);
 }//for j,m
}//MFFT
```

Такая функция позволяет выполнять множественное прямое БПФ (при D=1):

$$Y_{M\times J} = MFFT^{N}(X_{M\times J})$$

или обратное БПФ (при D=0):

$$Y_{M\times J} = iMFFT^{N}(X_{M\times J})$$

с комплексными векторами длиной N, являющимися элементами матрицы X[M][J], помещая результат в матрицу Y[M][J] комплексных векторов, используя при этом комплексный вектор W в качестве таблицы весовых коэффициентов W[0:N]. В ней в теле внутреннего цикла (по j) осуществляется одиночная операция БПФ над векторами, расположенными начиная с позиции mj=(m*J+j)*N в матрицах X и Y.

Помимо операции множественного БПФ для осуществления операции множественной быстрой свёртки потребуется ещё и операция покомпонентного умножения комплексных векторов, являющихся элементами соответствующих матриц размером M×J. Оформим такую операцию в виде следующей функции **MMul**:

```
void MMul(int M,int J,int N,
complex *X,complex *Y,complex *Z){
int m,j,mj,n;
 for (m=0; m<M; m++)
 for (j=0;j<J;j++){ mj=(m*J+j)*N;
 for (n=0; n<N; n++)
 Z[mj+n]= X[mj+n]*Y[mj+n];
 }//for j,m
}//MMul
```

Используя приведённые функции MFFF и MMul, представим теперь в виде функции MFSvrtka алгоритм исполнения операции быстрой свёртки для множества векторов, являющихся элементами соответствующих матриц:

```
void MFSvrtka(int M,int J,int N,
complex *X,complex *Y,complex *W,
complex *G, complex *Z){
 MFFT(M,J,N,1,X,Z,W); // Z=MFFT(X)
 MFFT(M,J,N,1,Y,G,W); // G=MFFT(Y)
 MMul(M,J,N,G,Z,G);  // G=MMul(G,Z)
 MFFT(M,J,N,0,G,Z,W); // Z=iMFFT(G)
}//MFSvrtka
```

Для этой функции в качестве исходных задаются комплексные вектора длиной N как элементы матриц X и Y размером M×J, а также подготовленная заранее таблица весовых коэффициентов Фурье W[0:N]. Результирующие вектора помещаются в матрицу Z[M][J][N], а матрица G[M][J][N] используется в качестве вспомогательной для сохранения промежуточных результатов преобразования.

Такой последовательный алгоритм выполнения операции множественной быстрой свёртки примем в качестве базового. Далее будем рассматривать различные варианты исполнения той же операции множественной свёртки в среде параллельных исполнителей. А также будем сравнивать каждый из них (по производительности) с этим базовым, чтобы выявить, какой вариант даёт максимальное ускорение в среде OpenCL.

3. Первоначальный вариант OpenCL-программы быстрой множественной свёртки

В качестве первоначального варианта (A) предложим такую OpenCL-программу, которая задействует M×J исполнителей OpenCL-среды для исполнения множественной операции свёртки параллельно сразу над всеми парами векторов, взятых на соответствующих позициях из матриц X и Y. При этом для обработки векторов применим тот же план действий, который ис-

пользовался в базовой программе, но будем выполнять его не последовательно, а параллельно для всех пар векторов.

Будем полагать, что в основной OpenCL-программе уже предусмотрены типовые подготовительные действия, необходимые для инициализации OpenCL-среды. Определена OpenCL-платформа и дескрипторы OpenCL-устройств, подготовлен OpenCL-контекст **cntxt** и в нём создан дескриптор очереди команд **cmndQ**, а также приготовлены все объекты **clX**, **clY**, **clZ**, **clG**, **clW** для представления в памяти OpenCL-устройства данных, подлежащих обработке, и особый объект **prgrm**, содержащий в откомпилированной форме код всех процедур ядра. (Как обеспечить все эти действия, было подробно описано ранее в пп. 2.3, 2.4, 2.5 в [3]).

Будем также считать, что в глобальной памяти OpenCL-устройства уже приготовлена таблица весовых коэффициентов Фурье, доступная как объект **clW** (типа cl_mem).

И далее представим процедуры ядра и функции OpenCL-программы, которые обеспечат весь комплекс действий по подготовке и запуску M×J параллельных вычислительных узлов OpenCL-среды для выполнения множественной операции быстрой свёртки.

```
void OpenCLMFSvrtka
(int M, int J, int N,complex *X,
complex *Y,complex *G,complex *Z){
 OpenCLMFFT(M,J,N,X,Z,1);
 OpenCLMFFT(M,J,N,Y,G,1);
 OpenCLMMul(M,J,N,G,Z,G);
 OpenCLMFFT(M,J,N,G,Z,0); }
//OpenCL_MFSvrtka
```

Функции OpenCLMFFT и OpenCLMMul, вызываемые здесь в теле основной функции OpenCLMFSvrtka, предназначены для исполнения в OpenCL-среде операций множественного БПФ и покомпонентного умножения, аналогичные тем, что были рассмотрены ранее в последовательной программе.

Функция **OpenCLMMul** сначала загружает исходные данные матриц Z и Y в память OpenCL-устройства и затем вызывает функцию **clMMul**, которая запускает на каждом вычислительном узле устройства процедуру ядра для исполнения операции покомпонентного умножения назначенной ей пары векторов.

```
void OpenCLMMul(int M,int J,int N,
complex *Z,complex *Y,complex *G){
// write Z matrix into clZ of device memory
cl_uint szC= sizeof(complex);
clEnqueueWriteBuffer(cmndQ,clZ,
 CL_TRUE,0,M*J*N*szC,Z,0,0,0);
// write Y matrix into clY of device memory
clEnqueueWriteBuffer(cmndQ,clY,
 CL_TRUE,0,M*J*N*szC,Y,0,0,0);
// perform clG=MMul(clZ,clY) in OpenCL-device
clMMul(M,J,N,&clZ,&clY,&clG);
// read the result from clG to the Host in matrix G
clEnqueueReadBuffer(cmndQ,clG,
 CL_TRUE,0,M*J*N*szC,G,0,0,0);
}//OpenCLMMul
void clMMul(int M, int J, int N,
cl_mem *pZ,cl_mem *pY,cl_mem *pG){
cl_uint szI= sizeof(int);
cl_uint szM= sizeof(cl_mem);
clSetKernelArg(knMM,0,szI,&M);
clSetKernelArg(knMM,1,szI,&J);
clSetKernelArg(knMM,2,szI,&N);
clSetKernelArg(knMM,3,szM,pZ);
clSetKernelArg(knMM,4,szM,pY);
clSetKernelArg(knMM,5,szM,pG);
size_t gWS[2]={M,J};cl_event evMM;
clEnqueueNDRangeKernel(cmndQ,
 knMM,2,NULL,gWS,NULL,0,0,&evMM);
clFinish(cmndQ);
}//clMMul
```

А после исполнения такой операции над всеми парами векторов сформированная в OpenCL-памяти результирующая матрица, представленная дескриптором clG, уже копируется в матрицу G основной памяти. Саму же операцию покомпонентного умножения элементов двух векторов в памяти OpenCL-устройства исполняет процедура ядра, которая запускается параллельно на всех M×J узлах OpenCL-устройства:

```
__kernel void kernMMul
( uint M, uint J, uint N,
const __global complex *Z,
 const __global complex *Y,
  __global complex *G ) {
 int m,j,mj, i;
 m= get_global_id(0);
 j= get_global_id(1);
 mj=(m*J+j)*N;
 for (i=0; i<N; i++)
 G[mj+i]= Z[mj+i]*Y[mj+i];
}//kernMMul
```

В OpenCL-программе процедура ядра **kernMMul** представлена своим дескриптором **knMM**, который должен быть предварительно проинициализирован после компиляции программы ядра, приготовленной на языке OpenCL (в файле “MFSvrtka.cl”):

```
knMM=clCreateKernel(prgrm,"kernMMul",0);
```

Аналогично следует проинициализировать и дескриптор **knMF**:

```
knMF=clCreateKernel(prgrm,"kernMFFT",0);
```

который будет представлять в OpenCL-программе процедуру ядра **kernMFFT**, предназначенную для исполнения вычислительными

узлами OpenCL-устройства множественной операции БПФ:

```
__kernel void kernMFFT
( uint M, uint J, uint N, uint D,
 const __global complex *X,
  __global complex *Y,
 const __global complex *W ) {
 int m,j,mj, i; complex XP[NP];
 int P=iLog2(N);
 m= get_global_id(0);
 j= get_global_id(1);
 mj=(m*J+j)*N;
//1. Бит-реверсная перестановка:
 copy_brvprst(N,P,&X[mj],XP);
//2.Основной цикл исполнения бабочек Фурье:
 FFTMainLoopP(N,P,XP,W,D);
if(D)// Нормализация (для обратного БПФ)
 for(i=0;i<N;i++) Y[mj+i]=XP[i];
else { float Norm=1.00/(float)N;
 for (i=0;i<N;i++)
 Y[mj+i]=cMScal(XP[i],Norm);
}//if D
}//kernMFFT
```

В этой процедуре используются две вспомогательные функции, которые оформлены отдельно. Одна из них **copy_brvprst** осуществляет бит-реверсное копирование исходного вектора, взятого с позиции mj матрицы X в вектор XP, который располагается в быстродействующей локальной (private) памяти исполнителя:

```
void copy_brvprst (int N, int L,
__global complex *X,complex *XP){
 unsigned int i,j;
 for (i=0; i<N; i++) {
 BRvrsVal(j,i,L); XP[j]=X[i];
 }// for i
}//copy_brvprst
```

А другая функция **FFTMainLoopP** реализует основной цикл исполнения операций “бабочек Фурье” над элементами вектора XP:

```
void FFTMainLoopP(uint N, uint P,
 complex *XP,__global complex *W,
 int D){ complex V;
int t,s,h,G,R,d,k,u,a,b;
s=1;h=2;G=1;R=N/2;d=N/2;
for (t=1; t<=P; t++) {
 for(k=0,u=0;k<G;k++,u=u+d){
 if(D) V=W[u];else V=W[N-u];
 for(i=0,a=k; i<R; i++,a=a+h)
 {b=a+s; BF(Y[a],Y[b],V);}
 }//for k
 h=h*2;s=s*2;G=G*2;R=R/2;d=d/2;
}//for t
}//FFTMainLoopP
```

А завершается процедура **kernMFFT** копированием полученного вектора XP на своё место в матрицу Y (с позиции mj) с нормализацией элементов для обратного БПФ (при D=0).

Для запуска процедуры ядра **kernMFFT** параллельно на всех M×J вычислительных узлах OpenCL-устройства в OpenCL-программе вызывается функция **clMFFT** :

```
void clMFFT(int M, int J, int N,
 cl_mem *pX,cl_mem *pY, int D ){
cl_uint szI= sizeof(int);
cl_uint szM= sizeof(cl_mem);
clSetKernelArg(knMF,0,szI,&M);
clSetKernelArg(knMF,1,szI,&J);
clSetKernelArg(knMF,2,szI,&N);
clSetKernelArg(knMF,3,szI,&D);
clSetKernelArg(knMF,4,szM,pX);
clSetKernelArg(knMM,5,szM,pY);
clSetKernelArg(knMM,6,szM,&clW);
size_t gWS[2]={M,J};cl_event evMF;
clEnqueueNDRangeKernel(cmndQ,
 knMF,2,NULL,gWS,NULL,0,0,&evMF);
clFinish(cmndQ);
}//clMFFT
```

После исполнения каждым узлом процедуры **kernMFFT** в памяти OpenCL-устройства в матрице, задаваемой указателем **pY**, формируется результат множественной операции БПФ над соответствующими векторами, взятыми из матрицы, задаваемой указателем **pX**.

Функция **clMFFT** вызывается из тела функции **OpenCLMFFT** после того, как та запишет заданную ей в качестве параметра матрицу **X** в память OpenCL-устройства, выделенную под дескриптор **clX**. А после такого вызова матрицу, полученную как результат множественной операции БПФ, функция **OpenCLMFFT** перекопирует из OpenCL-памяти, выделенной под дескриптор **clZ**, в память OpenCL-программы по месту, задаваемому параметром **Z**.

```
void OpenCLMFFT(int M,int J,int N,
 complex *X,complex *Z, int D) {
// write X matrix into clX of device memory
cl_uint szC= sizeof(complex);
clEnqueueWriteBuffer(cmndQ,clX,
 CL_TRUE,0,M*J*N*szC,X,0,0,0);
// perform clZ=MFFT(clX) in OpenCL-device
clMFFT(M,J,N,&clX,&clZ,D);
// read the result from clZ to the Host in matrix Z
clEnqueueReadBuffer(cmndQ,clZ,
 CL_TRUE,0,M*J*N*szC,Z,0,0,0);
}//OpenCLMFFT
```

Заметим, что функция **OpenCLMFFT** вызывается из основной функции **OpenCLMFSvrtka** аж 3 раза. Первый раз, чтобы получить в матрице Z результат множественного прямого БПФ, произведённого в OpenCL-среде над векторами матрицы X. Второй раз, чтобы получить во вспомогательной матрице G результат множественного

прямого БПФ над векторами матрицы Y. А третий раз, чтобы получить итоговую матрицу Z как результат множественного обратного БПФ над векторами вспомогательной матрицы G, изменённой после её покомпонентного перемножения с векторами матрицы Z.

Теперь попробуем оценить, насколько удаётся ускорить операцию множественной быстрой свёртки полученным вариантом (А) OpenCL-программы. Для этого сравним время исполнения ($\mathbf{T_{CL}}$) функции **OpenCLMFSvrtka** (в среде OpenCL с множеством параллельных исполнителей) со временем исполнения ($\mathbf{T_{CPU}}$) функции **MFSvrtka** (на одном процессоре) с теми же параметрами и вычислим коэффициент ускорения **K** как отношение $\mathbf{T_{CPU}/T_{CL}}$.

Программа с вызовами этих функций была разработана (на языке Си) как консольное приложение в MS Visual Studio. Она компоновалась и многократно прогонялась на различных OpenCL-платформах для матриц комплексных векторов одинарной точности различных размеров (M×J) и длин (N). Усреднённые показатели коэффициентов ускорения, полученные в результате таких прогонов, представлены в таблицах 1 и 2.

Таблица 1 демонстрирует показатели ускорения, полученные на аппаратной платформе, оснащённой процессором CPU Intel-i59400 с частотой 2.9 Ггц и встроенным графическим процессором GPU UHD 630 с частотой 350 МГц (будем называть её платформа «Intel»).

Таблица 1. Ускорение множественной свёртки OpenCL-программой варианта А на платформе Intel

M×J: N	**2×2**	**5×5**	**10 ×10**	**20 ×20**	**50 ×50**	**100 ×100**	**200 ×200**
32	0.148	0.841	3.405	11.920	**29.990**	24.522	22.771
64	0.273	1.501	5.206	16.855	26.474	22.070	20.911
128	0.360	2.315	8.192	**22.874**	22.561	19.963	19.905
256	0.468	2.649	10.749	24.742	21.691	19.392	20.897
512	0.548	2.964	10.244	13.163	4.698	4.545	---
1024	0.600	3.539	10.498	10.034	4.396	3.970	---
2048	0.613	2.855	9.664	8.801	3.904	---	---
4096	0.641	3.012	9.181	7.070	3.944	---	---
8192	0.670	3.101	8.320	6.311	---	---	---
16384	0.645	3.229	6.590	5.992	---	---	---
32768	0.648	2.635	5.764	5.611	---	---	---
65536	0.661	2.199	6.379	---	---	---	---

Замечание: символы --- в таблице 1означают, что для таких значений параметров программа не может быть корректно исполнена (например, из-за недостатка памяти).

А таблица 2 представляет показатели ускорения, полученные на аппаратной платформе, содержащей процессор Intel i3-2100 с частотой 3.1 Ггц и специализированную видеокарту NVidia GeForce 1050ti с частотой 1392 Мгц (будем называть её платформа «NVidia»).

Таблица 2. Ускорение множественной свёртки OpenCLпрограммой варианта А на платформе NVidia

M×J: N	**2×2**	**5×5**	**10 ×10**	**20 ×20**	**50 ×50**	**100 ×100**	**200 ×200**
32	0.116	0.787	2.881	9.414	30.250	32.962	36.077
64	0.337	1.802	6.281	17.694	34.965	40.192	42.344
128	0.539	3.640	15.920	31.400	42.473	48.575	49.209
256	0.727	4.955	11.526	31.545	54.953	52.125	52.175
512	1.206	6.438	16.857	36.950	64.714	60.089	58.099
1024	1.429	6.716	18.811	44.194	66.867	64.929	61.224
2048	1.526	7.287	24.404	47.576	77.568	69.470	---
4096	1.499	7.874	23.853	56.910	82.800	75.480	---
8192	1.454	8.210	26.761	59.489	**84.016**	---	---
16384	1.499	8.515	26.673	77.436	**100.41**	---	---
32768	1.395	8.121	24.688	65.389	---	---	---
65536	1.168	6.290	21.880	33.502	---	---	---

Заметим, что в этом варианте А программы множественной свёртки был заимствован не самый оптимальный вариант реализации операции множественного БПФ, который был представлен ранее в [7, п.4]. Но сравнивая показатели ускорения операции множественного БПФ этого варианта В, представленные в таблицах 7 и 3 в [7], с показателями ускорения операции множественной свёртки из таблиц 1 и 2, можно заметить, что и на платформе “Intel”, и на платформе NVidia почти для всех параметров M,J,N операцию множественной свёртки не удалось ускорить лучше, чем саму операцию множественного БПФ. Так, например, на платформе NVidia на параметрах M×J=50×50 N=8192 для операции множественного БПФ удалось достичь ускорения в 125 раз, а для операции множественной свёртки – лишь в 84 раза.

Возникает вопрос, а можно ли вообще ускорить по технологии OpenCL операцию быстрой свёртки в лучшей степени, чем саму операцию БПФ, которая применяется для её исполнения? Оказывается, можно. Рассмотрим следующие варианты OpenCL-программы, которые позволяют этого добиться.

4. Варианты оптимизации OpenCL-программы быстрой множественной свёртки

Заметим, что в OpenCL-программе варианта А приходится многократно передавать блоки данных, представляющие матрицы комплексных чисел размером M×J×N, из основной памяти в память OpenCL-устройства и обратно. При каждом вызове функции **OpenCLMFFT** такая передача выполняется дважды, а при вызове функции **OpenCLMMul** ─ трижды. Итого для одного исполнения функции **OpenCLMFSvrtka** потребуется аж 9 таких передач блоков данных.

Попытаемся модифицировать функцию множественной свёртки так, чтобы минимизировать

количество таких передач. Для этого все матрицы, в которых формируются промежуточные результаты, будем просто оставлять в памяти OpenCL-устройства, не копируя их в основную память. И затем сразу же их использовать при последующей обработке процедурами ядра.

Выразим такие изменения в OpenCL-программе, оформив другую функцию для исполнения множественной свёртки:

```
void OpenCLMFSvrtkaB
(int M, int J, int N,complex *X,
complex *Y,complex *G,complex *Z){
// write X matrix into clX of device memory
cl_uint szC= sizeof(complex);
clEnqueueWriteBuffer(cmndQ,clX,
 CL_TRUE,0,M*J*N*szC,X,0,0,0);
// write Y matrix into clY of device memory
clEnqueueWriteBuffer(cmndQ,clY,
 CL_TRUE,0,M*J*N*szC,Y,0,0,0);
// perform clZ=MFFT(clX) in OpenCL-device
clMFFT(M,J,N,&clX,&clZ,1);
// perform clG=MFFT(clY) in OpenCL-device
clMFFT(M,J,N,&clY,&clG,1);
// perform clG=MMul(clG,clZ) in OpenCL-device
clMMul(M,J,N,&clG,&clZ,&clG);
// perform clZ=iMFFT(clG) in OpenCL-device
clMFFT(M,J,N,&clG,&clZ,0);
// read the result from clZ to the Host in matrix Z
clEnqueueReadBuffer(cmndQ,clZ,
 CL_TRUE,0,M*J*N*szC,Z,0,0,0);
//OpenCL_MFSvrtkaB
```

В результате такой модификации количество передач матричных блоков данных для одной операции свёртки сократится с девяти до трёх. Посмотрим, как это отразится на показателях, характеризующих ускорение операции множественной свёртки в среде OpenCL.

Результаты полученного нового варианта (B) OpenCL-программы представлены в таблицах 3 и 4 (для платформ «Intel» и «NVidia»).

Можно отметить, что показатели ускорения в этих таблицах почти для всех параметров M,J,N заметно улучшились (по сравнению с вариантом A). Причём теперь коэффициенты ускорения операции множественной свёртки во многих случаях даже превосходят коэффициенты ускорения множественной операции БПФ (из таблиц 7 и 3 в [7]) для данных тех же размеров. Так, например, на параметрах M×J=50×50 N=8192 для платформы NVidia коэффициент ускорения множественной операции свёртки вырос до значения **151** и тем самым превзошёл значение 125, которое было отмечено для операции множественного БПФ (в таблице 3 в [7]).

Таблица 3. Ускорение множественной свёртки OpenCL-программой варианта B на платформе Intel

M×J:	**2×2**	**5×5**	**10**	**20**	**50**	**100**	**200**
N			**×10**	**×20**	**×50**	**×100**	**×200**
32	0.153	0.919	3.733	13.372	37.219	39.701	**40.897**
64	0.282	1.586	6.002	21.210	38.644	32.144	30.269
128	0.355	2.398	8.755	26.735	30.126	26.496	26.554
256	0.468	2.649	10.749	24.742	21.691	19.392	28.145
512	0.549	2.943	10.547	14.664	4.757	4.706	---
1024	0.592	3.551	10.865	10.882	4.534	4.094	---
2048	0.632	2.991	8.946	9.431	3.995	---	---
4096	0.615	3.049	9.558	7.894	4.025	---	---
8192	0.661	3.114	8.433	6.253	---	---	---
16384	0.643	3.273	6.721	6.206	---	---	---
32768	0.649	2.661	5.930	5.961	---	---	---
65536	0.660	2.237	6.548	---	---	---	---

Таблица 4. Ускорение множественной свёртки OpenCLпрограммой варианта B на платформе NVidia

M×J: N	**2×2**	**5×5**	**10 ×10**	**20 ×20**	**50 ×50**	**100 ×100**	**200 ×200**
32	0.233	1.308	5.000	17.063	65.154	65.923	74.053
64	0.500	2.646	11.167	35.389	71.179	75.075	77.631
128	0.710	4.440	26.534	52.333	93.440	80.958	90.250
256	0.941	6.812	19.909	57.833	104.41	94.773	99.606
512	1.708	8.822	24.842	67.182	104.10	107.89	104.74
1024	1.818	8.569	26.237	72.318	132.06	106.90	112.54
2048	1.723	8.132	31.888	80.052	139.44	119.99	---
4096	1.578	8.574	29.346	83.549	137.85	129.63	---
8192	1.490	8.794	31.208	86.613	**151.05**	---	---
16384	1.517	9.042	31.106	111.78	**174.67**	---	---
32768	1.416	8.488	27.812	90.039	---	---	---
65536	1.175	6.455	23.852	38.298	---	---	---

Опробуем ещё один возможный путь оптимизации OpenCL-программы. Он связан с минимизацией накладных расходов, требующихся на запуск процедур ядра. В предыдущем варианте B требовалось обеспечить 4 таких запуска. Попробуем сосредоточить все действия, которые надлежит исполнить каждым вычислительным узлом OpenCL-устройства, в одной единственной процедуре ядра. Такая процедура будет запускаться один раз (на всех узлах сразу) и должна будет исполнить весь комплекс действий с назначенными ей векторами из заданных матриц, требующийся для получения свёртки.

```
__kernel void kernMFSvrtka
( uint M, uint J, uint N,
 const __global complex *X,
 const __global complex *W,
  __global complex *Y,
  __global complex *Z ) {
unsigned int m,j,mj, i,q;
complex V, T;
complex XP[NP]; complex YP[NP];
const __global complex *VX;
__global complex *VY,*VZ;
int P=iLog2(N);
m=get_global_id(0);
j=get_global_id(1); mj=(m*J+j)*N;

VX=&X[mj]; VY=&Y[mj]; VZ=&Z[mj];
//1. Прямое БПФ: XP=FFT(VX); YP=FFT(VY);
```

```
//1a. Бит-реверсное копир. VX=>XP; VY=>YP;
for(i=0;i<N;i++){ BRvrsVal(q,i,P);
 XP[q]=VX[i]; YP[q]=VY[i];}//for i
//1b.Основной цикл исполнения бабочек Фурье:
int t,s,h,G,R,d,k,u,a,b;
s=1;h=2;G=1;R=N/2;d=N/2;
for (t=1; t<=P; t++) {
for(k=0,u=0;k<G;k++,u=u+d){V=W[u];
 for(i=0,a=k;i<R;i++,a=a+h){b=a+s;
 BF(XP[a],XP[b],V);
 BF(YP[a],YP[b],V); }//for i
}//for k
h=h*2;s=s*2;G=G*2;R=R/2;d=d/2;
}//for t
//2. Поэлем. перемножение: XP=MMul(XP,YP)
for(i=0;i<N;i++)XP[i]=XP[i]*YP[i];
//3. Обратное БПФ: VZ=iFFT(XP);
//3a. Бит-реверсная перестановка вектора XP:
for(i=0;i<N;i++){ BRvrsVal(q,i,P);
 if(q>i)//XP[q]<=>XP[i];
 {T=XP[q]; XP[q]=XP[i]; XP[i]=T;}
}// for i
//3b.Основной цикл исполнения бабочек Фурье:
FFT_MainLoopP(N,P,XP,W,0);
//3c Копирование с нормализацией XP=>VX :
float Norm=1.00/(float)N;
for (i=0;i<N;i++)
 VZ[i]=cMScal(XP[i],Norm);
}//for i
}//kernMFSvrtka
```

В этой процедуре применён ещё один приём оптимизации: выполняется совместное прямое БПФ сразу для двух векторов VX и VY, взятых из исходных матриц X и Y (на позиции mj). Это позволяет не вычислять дважды индексы в цикле бит-реверсного копирования и многочисленные параметры в цикле исполнения операций «бабочек Фурье».

Для новой процедуры ядра **kernMFSvrtka** проинициализируем дескриптор, представляющий её в OpenCL-программе:

```
knMS=clCreateKernel(prgrm,"kernMFSvrtka",0);
```

И составим функцию, которая будет выполнять операцию множественной свёртки, запуская лишь эту единственную процедуру ядра на M×J вычислительных узлах OpenCL-среды:

```
void OpenCLMFSvrtkaC
(int M, int J, int N,complex *X,
complex *Y,complex *G,complex *Z){
// write X matrix into clX of device memory
clEnqueueWriteBuffer(cmndQ,clX,
 CL_TRUE,0,M*J*N*szC,X,0,0,0);
// write Y matrix into clY of device memory
clEnqueueWriteBuffer(cmndQ,clY,
 CL_TRUE,0,M*J*N*szC,Y,0,0,0);
clSetKernelArg(knMS,0,szI,&M);
clSetKernelArg(knMS,1,szI,&J);
clSetKernelArg(knMS,2,szI,&N);
clSetKernelArg(knMS,3,szM,&clX);
clSetKernelArg(knMS,4,szM,&clW);
clSetKernelArg(knMS,5,szM,&clY);
clSetKernelArg(knMS,6,szM,&clZ);
size_t gWS[2]={M,J};cl_event evMS;
clEnqueueNDRangeKernel(cmndQ,
 knMS,2,NULL,gWS,NULL,0,0,&evMS);
clFinish(cmndQ);
// read the result from clZ to the Host in matrix Z
clEnqueueReadBuffer(cmndQ,clZ,
 CL_TRUE,0,M*J*N*szC,Z,0,0,0);
}//OpenCL_MFSvrtkaC
```

В результате такой оптимизации получим ещё один вариант (C) OpenCL-программы для выполнения множественной свёртки в среде OpenCL. Показатели ускорения этой операции (вариантом C) для платформ «Intel» и «NVidia» представим в таблицах 5 и 6 соответственно.

Таблица 5. Ускорение множественной свёртки OpenCL-программой варианта C на платформе Intel

M×J: N	2×2	5×5	10 ×10	20 ×20	50 ×50	100 ×100	200 ×200
32	0.166	0.978	3.890	12.450	38.678	42.702	46.873
64	0.291	1.716	6.074	19.584	36.668	41.656	45.263
128	0.388	2.457	8.519	25.621	38.743	41.033	46.240
256	0.521	2.973	11.210	29.956	16.262	13.731	15.549
512	0.613	3.217	11.033	12.008	4.880	4.862	---
1024	0.659	3.709	11.132	10.132	4.572	4.241	---
2048	0.624	3.131	9.648	8.322	4.023	---	---
4096	0.701	2.781	9.610	7.158	4.025	---	---
8192	0.656	3.244	7.718	6.642	---	---	---
16384	0.688	3.131	6.326	6.100	---	---	---
32768	0.658	2.426	6.169	5.596	---	---	---
65536	0.680	2.156	6.065	---	---	---	---

Таблица 6. Ускорение множественной свёртки OpenCLпрограммой варианта C на платформе NVidia

M×J: N	2×2	5×5	10 ×10	20 ×20	50 ×50	100 ×100	200 ×200
32	0.400	2.576	8.095	27.300	77.000	81.619	93.800
64	0.814	3.943	16.080	37.471	79.720	88.422	98.803
128	1.000	7.114	31.841	52.333	116.80	107.95	121.13
256	1.311	8.385	24.333	69.400	116.01	115.83	117.39
512	1.864	9.528	27.765	67.182	126.02	131.86	128.00
1024	1.818	8.283	27.694	75.762	135.45	139.26	131.29
2048	1.513	8.502	32.319	78.349	148.26	140.24	---
4096	1.596	8.929	31.300	84.426	146.57	149.87	---
8192	1.480	10.452	35.873	103.49	**154.82**	---	---
16384	1.530	9.196	31.495	111.15	---	---	---
32768	1.781	10.298	31.106	65.087	---	---	---
65536	1.205	6.635	23.705	---	---	---	---

Как видно из приведённых таблиц, для многих наборов параметров (M,J,N) вариант C OpenCL-программы обеспечивает и на платформе Intel, и на платформе NVidia наилучшие показатели ускорения среди всех уже рассмотренных нами вариантов (A,B,C).

5. Оптимизация множественной свёртки комплексных векторов большой длины

До сих пор для выполнения множественной свёртки применялся простой, но не самый оптимальный вариант множественной операции БПФ, описанный в [7,п.4] как вариант B. Ранее был предложен более оптимальный способ исполнения операции множественного БПФ в среде OpenCL, описанный в [7,п.5] как вариант C. Он позволяет существенно ускорить такую операцию для комплексных векторов большой длины. Попробуем применить такой же вариант ускорения прямого и обратного множественного БПФ для дальнейшей оптимизации множественной свёртки.

Процедуры ядра множественного БПФ, представленные в [7,п.5] модифицируем так, чтобы приспособить их для исполнения как прямого (при D=1), так и обратного БПФ (при D=0) ансамблем из N×M×J параллельных исполнителей. Для этого переделаем процедуру бит-реверсного копирования элемента:

```
__kernel void kernMFFTbp
( uint N, uint L,
 const __global complex *X,
  __global complex *Y,
 uint M, uint J ) {
 uint m,j,mj, k,i;
 i= get_global_id(0);
 m= get_global_id(1);
 j= get_global_id(2);
 mj=(m*J+j)*N;
 BRvrsVal(k,i,L);
 Y[mj+k]= X[mj+i];
}//kernMFFTbp
```

А также процедуру исполнения операции «бабочки Фурье» для пары элементов:

```
__kernel void kernMFFTbf
( uint t, uint N,
 __global complex *Y,
 __global complex *W,
 uint M, uint J, uint D) {
 uint gm,gj,mj, gi; complex V;
 uint s,h,G,R,k,u,j,a,b;
 gi= get_global_id(0);
 gm= get_global_id(1);
 gj= get_global_id(2);
 mj=(gm*J+gj)*N;
G=1<<(t-1);R=N>>t;// G=2^(t-1),R=2^(P-t);
k=gi&(G-1);j=gi>>(t-1);//k=gi%G,j=gi/G
 s=G;h=1<<t; a=k+j*h;b=a+s; u=R*k;
 V=D? W[u]:W[N-u] /*Conj(W[u])*/;
 BF(Y[mj+a],Y[mj+b],V);}
}//kernMFFTbf
```

И добавим процедуру нормализации элемента:

```
__kernel void kernMFFTnm
( uint N, uint M, uint J,
__global complex *X, float Norm){
 uint i= get_global_id(0);
 uint m= get_global_id(1);
 uint j= get_global_id(2);
 uint mj=(m*J+j)*N;
 X[mj+i]=cMScal(X[mj+i],Norm);
}//kernMFFTnm
```

В OpenCL-программе проинициализируем дескрипторы этих процедур:

```
knMp=clCreateKernel(prgrm,"kernMFFTbp",0);
knMb=clCreateKernel(prgrm,"kernMFFTbf",0);
knMn=clCreateKernel(prgrm,"kernMFFTnm",0);
```

И предусмотрим функцию, которая будет запускать эти процедуры ядра параллельно на N×M×J вычислительных узлах OpenCL-среды, чтобы выполнить операцию множественного БПФ сразу над всеми векторами, уложенными в памяти OpenCL-устройства в матрицы, заданные в параметрах функции указателями на дескрипторы (pX,pY).

```
void clMFFTD(int M, int J, int N,
 cl_mem *pX,cl_mem *pY, int D ){
cl_uint szI= sizeof(int);
cl_uint szM= sizeof(cl_mem);
cl_uint szF=sizeof(float);
cl_event evMp,evMb,evMn;
size_t gWS[3]={N,M,J};
uint P=iLog2(N);
// 1. Bit-reverse copyng X[m][j] => Y[m][j]
clSetKernelArg(knMp,0,szI,&N);
clSetKernelArg(knMp,1,szI,&P);
clSetKernelArg(knMp,2,szM,pX);
clSetKernelArg(knMp,3,szM,pY);
clSetKernelArg(knMp,4,szI,&M);
clSetKernelArg(knMp,5,szI,&J);
clEnqueueNDRangeKernel(cmndQ,
 knMp,3,NULL,gWS,NULL,0,0,&evMp);
clFinish(cmndQ);
// 2. Main Loop for execution butterfly operations:
clSetKernelArg(knMb,1,szI,&N);
clSetKernelArg(knMb,2,szM,pY);
clSetKernelArg(knMb,3,szM,&clW);
clSetKernelArg(knMb,4,szI,&M);
clSetKernelArg(knMb,5,szI,&J);
clSetKernelArg(knMb,6,szI,&D);
gWS[0]= N/2;
for (int t=1; t<=P; t++) {
 clSetKernelArg(knMb,0,szI,&t);
 clEnqueueNDRangeKernel(cmndQ,
 knMF,3,NULL,gWS,NULL,0,0,&evMb);
 clWaitForEvents(1,&evMb);
}//for t
clFinish(cmndQ);
// 3. Normalization (for InDirect FFT)
```

```
if(!D) { gWS[0]=N;
 float Norm=1.00/(float)N;
 clSetKernelArg(knMn,0,szI,&N);
 clSetKernelArg(knMn,1,szI,&M);
 clSetKernelArg(knMn,2,szI,&J);
 clSetKernelArg(knMn,3,szM,pY);
 clSetKernelArg(knMn,4,szF,&Norm);
 clEnqueueNDRangeKernel(cmndQ,
 knMn,3,NULL,gWS,NULL,0,0,&evMn);
 clFinish(cmndQ); }//if
}//clMFFTD
```

Используя её, составим для нового варианта (D) OpenCL-программы функцию выполнения множественного БПФ над M×J векторами, уложенными в заданные матрицы:

```
void OpenCLMFFTD(int M,int J,int N,
 complex *X,complex *Z, int D) {
// write X matrix into clX of device memory
clEnqueueWriteBuffer(cmndQ,clX,
 CL_TRUE,0,M*J*N*szC,X,0,0,0);
// perform clZ=MFFT(clX) in OpenCL-device
clMFFTD(M,J,N,&clX,&clZ,D);
// read the result from clZ to the Host in matrix Z
clEnqueueReadBuffer(cmndQ,clZ,
 CL_TRUE,0,M*J*N*szC,Z,0,0,0);
}//OpenCLMFFTD
```

Такая функция **OpenCLMFFTD** записывает исходную матрицу векторов, заданную как параметр X, из основной памяти в память OpenCL-устройства, затем вызывает функцию **clMFFTD** для исполнения на устройстве множественного БПФ сразу над всеми векторами, а после копирует полученный результат из памяти устройства в основную память по месту расположения матрицы, заданной параметром Z.

Аналогичную оптимизацию проделаем и для множественной операции покомпонентного умножения, используя для её исполнения в OpenCL-среде не M×J, а N×M×J вычислителей.

Для этого определим новую процедуру ядра, в которой обработка всех N элементов векторов выполняется не одним исполнителем в последовательном цикле, а параллельно сразу N исполнителями, каждый из которых получает результат перемножения для своего элемента:

```
__kernel void kernMMulD
( uint M, uint J, uint N,
 const __global complex *Z,
 const __global complex *Y,
  __global complex *G ) {
 int i= get_global_id(0);
 int m= get_global_id(1);
 int j= get_global_id(2);
 int mj=(m*J+j)*N;
 G[mj+i]= Z[mj+i]*Y[mj+i];
}//kernMMulD
```

В OpenCL-программе проинициализируем дескриптор новой процедуры и модифицируем соответствующие функции:

```
knMm=clCreateKernel(prgrm,"kernMMulD",0);
```

```
void clMMulD(int M, int J, int N,
cl_mem *pZ,cl_mem *pY,cl_mem *pG){
cl_uint szI= sizeof(int);
cl_uint szM= sizeof(cl_mem);
clSetKernelArg(knMm,0,szI,&N);
clSetKernelArg(knMm,1,szI,&M);
clSetKernelArg(knMm,2,szI,&J);
clSetKernelArg(knMm,3,szM,pZ);
clSetKernelArg(knMm,4,szM,pY);
clSetKernelArg(knMm,5,szM,pG);
size_t gWS[3]={N,M,J};
cl_event evMm;
clEnqueueNDRangeKernel(cmndQ,
 knMm,3,NULL,gWS,NULL,0,0,&evMm);
clFinish(cmndQ);
}//clMMulD
```

```
void OpenCLMMulD(int M,int J,int N,
complex *Z,complex *Y,complex *G){
// write Z matrix into clZ of device memory
cl_uint szC= sizeof(complex);
clEnqueueWriteBuffer(cmndQ,clZ,
 CL_TRUE,0,M*J*N*szC,Z,0,0,0);
// write Y matrix into clY of device memory
clEnqueueWriteBuffer(cmndQ,clY,
 CL_TRUE,0,M*J*N*szC,Y,0,0,0);
// perform clG=MMul(clZ,clY) in OpenCL-device
clMMulD(M,J,N,&clZ,&clY,&clG);
// read the result from clG to the Host in matrix G
clEnqueueReadBuffer(cmndQ,clG,
 CL_TRUE,0,M*J*N*szC,G,0,0,0);
}//OpenCLMMulD
```

Используя оптимизированные варианты функций **OpenCLMFFTD** и **OpenCLMMulD**, составим новый вариант (D) функции для исполнения операции множественной свёртки в среде OpenCL:

```
void OpenCLMFSvrtkaD
(int M, int J, int N,complex *X,
complex *Y,complex *G,complex *Z){
 OpenCLMFFTD(M,J,N,X,Z,1);
 OpenCLMFFTD(M,J,N,Y,G,1);
 OpenCLMMulD(M,J,N,G,Z,G);
 OpenCLMFFTD(M,J,N,G,Z,0);
}//OpenCL_MFSvrtkaD
```

В результате проведённой оптимизации получим вариант (D) OpenCL-программы для выполнения множественной свёртки в среде OpenCL, который позволяет значительно ускорить такую операцию для векторов большой длины. Это подтверждается улучшенными коэффициентами ускорения варианта D, представленными в таблицах 7 и 8 для платформ «Intel» и «NVidia» соответственно, по сравнению с теми

же коэффициентами варианта A (из таблиц 1 и 2), по крайней мере, для всех наборов параметров M,J,N при N≥512.

Таблица 7. Ускорение множественной свёртки OpenCL-программой варианта D на платформе Intel

M×J: N	2×2	5×5	10 ×10	20 ×20	50 ×50	100 ×100	200 ×200
32	0.043	0.266	1.077	3.978	16.483	26.564	22.720
64	0.091	0.548	2.101	7.148	23.940	28.269	21.207
128	0.172	1.165	4.053	12.510	28.755	27.373	22.960
256	0.354	2.201	7.887	19.826	33.537	25.227	24.365
512	0.696	4.088	13.143	29.752	32.775	29.392	---
1024	1.405	8.227	22.293	33.875	30.287	25.662	---
2048	2.204	12.200	27.474	38.430	31.317	---	---
4096	5.027	16.844	37.649	32.421	30.034	---	---
8192	7.178	37.084	43.632	30.501	---	---	---
16384	17.38	49.275	31.167	28.455	---	---	---
32768	28.33	**48.156**	37.050	34.616	---	---	---
65536	37.28	39.669	34.137	---	---	---	---

Заметим, однако, что на платформе «NVidia» на некоторых наборах параметров даже при больших значениях N показатели варианта D уступают показателям варианта B (и тем более C). Так, например, для M×J=50×50 и N=8192 OpenCL-программа варианта D даёт коэффициент ускорения 93.440, что лучше, чем значение 84.016 варианта A, но хуже значения 151.05 варианта B (и значения 154.82 варианта C).

Таблица 8. Ускорение множественной свёртки OpenCLпрограммой варианта D на платформе NVidia

M×J: N	2×2	5×5	10 ×10	20 ×20	50 ×50	100 ×100	200 ×200
32	0.063	0.429	1.441	5.151	18.822	30.607	37.520
64	0.164	0.954	3.865	9.800	28.884	38.260	44.361
128	0.277	1.932	13.267	22.429	46.720	51.132	52.490
256	0.640	4.192	14.600	38.556	49.719	54.868	57.667
512	1.367	5.955	24.842	43.471	63.011	65.027	62.493
1024	2.963	14.200	35.607	54.862	74.402	65.736	64.772
2048	4.807	28.774	56.943	68.193	85.494	76.064	---
4096	10.46	40.300	64.561	83.549	84.556	83.597	---
8192	23.47	55.851	79.674	90.359	**93.440**	---	---
16384	28.23	68.284	90.944	121.36	114.12	---	---
32768	44.67	78.675	105.53	113.57	---	---	---
65536	54.03	105.05	113.89	**131.30**	---	---	---

Попробуем улучшить OpenCL-программу варианта D, применив тот же приём оптимизации, какой был применён при превращении программы варианта A в программу варианта B. Для этого постараемся минимизировать количество передач матричных блоков из основной памяти в память OpenCL-устройства и обратно при исполнении операции множественной свёртки в среде OpenCL. Получим следующий вариант (E) функции множественной свёртки:

```
void OpenCLMFSvrtkaE
(int M, int J, int N,complex *X,
complex *Y,complex *G,complex *Z){
// write X matrix into clX of device memory
cl_uint szC= sizeof(complex);
clEnqueueWriteBuffer(cmndQ,clX,
 CL_TRUE,0,M*J*N*szC,X,0,0,0);
// write Y matrix into clY of device memory
clEnqueueWriteBuffer(cmndQ,clY,
 CL_TRUE,0,M*J*N*szC,Y,0,0,0);
// perform clZ=MFFT(clX) in OpenCL-device
clMFFTD(M,J,N,&clX,&clZ,1);
// perform clG=MFFT(clY) in OpenCL-device
clMFFTD(M,J,N,&clY,&clG,1);
// perform clG=MMul(clZ,clY) in OpenCL-device
clMMulD(M,J,N,&clZ,&clY,&clG);
// perform clZ=iMFFT(clG) in OpenCL-device
clMFFTD(M,J,N,&clG,&clZ,0);
// read the result from clZ to the Host in matrix Z
clEnqueueReadBuffer(cmndQ,clZ,
 CL_TRUE,0,M*J*N*szC,Z,0,0,0);
}//OpenCL_MFSvrtkaE
```

Коэффициенты ускорения, полученные в результате многократных прогонов OpenCL-программы варианта E, применяющей функцию **OpenCLMFSvrtkaE**, представлены в таблицах 9 (для «Intel») и 10 (для «NVidia»).

Таблица 9. Ускорение множественной свёртки OpenCL-программой варианта E на платформе Intel

M×J: N	2×2	5×5	10 ×10	20 ×20	50 ×50	100 ×100	200 ×200
32	0.045	0.268	1.087	3.795	17.264	32.846	30.792
64	0.093	0.543	2.131	6.819	27.266	37.311	26.393
128	0.178	1.176	4.144	12.470	35.177	33.262	28.811
256	0.357	2.217	7.497	21.544	41.577	30.334	29.431
512	0.704	4.131	12.529	32.044	40.671	35.148	---
1024	1.375	8.698	23.303	38.921	35.608	33.206	---
2048	2.301	12.729	33.158	45.598	36.581	---	---
4096	4.976	21.606	44.070	37.425	25.448	---	---
8192	9.727	39.396	51.513	33.221	---	---	---
16384	17.89	56.034	35.912	34.431	---	---	---
32768	29.57	**56.226**	41.890	38.317	---	---	---
65536	41.44	43.630	37.650	---	---	---	---

Таблица 10. Ускорение множественной свёртки OpenCLпрограммой варианта E на платформе NVidia

M×J: N	2×2	5×5	10 ×10	20 ×20	50 ×50	100 ×100	200 ×200
32	0.081	0.462	2.179	6.825	38.500	53.563	70.350
64	0.174	1.030	4.517	14.814	51.103	71.054	80.506
128	0.315	2.484	15.920	39.250	66.743	75.309	104.98
256	0.800	5.450	19.909	49.571	80.315	99.286	86.500
512	1.464	8.822	39.333	73.900	126.02	115.78	114.41
1024	3.478	17.138	55.389	93.588	124.29	125.99	124.38
2048	6.214	38.697	95.664	122.75	162.68	132.78	---
4096	13.07	58.976	107.60	145.44	147.75	143.83	---
8192	25.61	80.769	131.46	163.89	**174.72**	---	---
16384	36.70	106.58	158.36	210.86	206.63	---	---
32768	60.58	143.99	185.47	201.98	---	---	---
65536	98.83	173.78	201.76	**229.01**	---	---	---

Они показывают, что вариант E OpenCL-программы во многих случаях обеспечивает более высокие коэффициенты ускорения. В частности, для набора параметров M×J=50×50 и

N=8192 на платформе «NVidia» коэффициент ускорения варианта **E** достигает значения **174.72**, что больше не только значения 151.05 варианта B, но и значения 154.82 варианта C.

6. Сравнение коэффициентов ускорения OpenCL-программ различных вариантов

Для выбора оптимального способа ускорения операции множественной свёртки с применением технологии OpenCL сравним показатели ускорения различных вариантов, рассмотренных ранее OpenCL-программ. Для этого соберём полученные на каждой платформе («Intel» и «NVidia») показатели ускорения в единые сводные таблицы (см. таблицы 11, 12).

В каждой строке этих таблиц приводятся показатели ускорения операции множественной свёртки, полученные для заданного набора параметров (M, J, N) пятью различными вариантами программ (A, B, C, D, E), разработанных с применением технологии OpenCL. При этом отмечается (жирным шрифтом) тот вариант, который обеспечивает для заданных значений параметров наилучшее ускорение операции.

Таблица 11. Ускорение множественной свёртки OpenCL-программами на платформе Intel

N	M×J	A	B	C	D	E
32	2×2	0.148	0.153	**0.166**	0.043	0.045
	5×5	0.841	0.919	**0.978**	0.266	0.268
	10×10	3.405	3.733	**3.890**	1.077	1.087
	20×20	11.920	**13.372**	12.450	3.978	3.795
	50×50	29.990	37.219	**38.678**	16.483	17.264
	100×100	24.522	39.701	**42.702**	26.564	32.846
	200×200	22.771	40.897	**46.873**	22.720	30.792
64	2×2	0.273	0.282	**0.291**	0.091	0.093
	5×5	1.501	1.586	**1.716**	0.548	0.543
	10×10	5.206	6.002	**6.074**	2.101	2.131
	20×20	16.855	**21.210**	19.584	7.148	6.819
	50×50	26.474	38.644	**36.668**	23.940	27.266
	100×100	22.070	32.144	**41.656**	28.269	37.311
	200×200	20.911	30.269	**45.263**	21.207	26.393
128	2×2	0.360	0.355	**0.388**	0.172	0.178
	5×5	2.315	2.398	**2.457**	1.165	1.176
	10×10	8.192	**8.755**	8.519	4.053	4.144
	20×20	22.874	**26.735**	25.621	12.510	12.470
	50×50	22.561	30.126	**38.743**	28.755	35.177
	100×100	19.963	26.496	**41.033**	27.373	33.262
	200×200	19.905	26.554	**46.240**	23.861	28.811
256	2×2	0.468	0.477	**0.521**	0.354	0.357
	5×5	2.649	2.684	**2.973**	2.201	2.217
	10×10	10.749	11.080	**11.210**	7.887	7.497
	20×20	24.742	**31.281**	29.956	19.826	21.544
	50×50	21.691	27.604	16.262	33.537	**41.577**
	100×100	19.392	23.968	13.731	25.227	**30.334**
	200×200	20.897	28.145	15.549	24.365	**29.431**
512	2×2	0.548	0.549	0.613	0.696	**0.704**
	5×5	2.964	2.943	3.217	4.088	**4.131**
	10×10	10.244	10.547	11.033	**13.143**	12.529
	20×20	13.163	14.664	12.008	29.752	**32.044**
	50×50	4.698	4.757	4.880	32.775	**40.671**
	100×100	4.545	4.706	4.862	29.392	**35.148**
1024	2×2	0.600	0.592	0.659	**1.405**	1.375
	5×5	3.539	3.551	3.709	8.227	**8.698**
	10×10	10.498	10.865	11.132	22.293	**23.303**
	20×20	10.034	10.882	10.132	33.875	**38.921**
	50×50	4.396	4.534	4.572	30.287	**35.608**
	100×100	3.970	4.094	4.241	28.413	**33.206**
2048	2×2	0.613	0.632	0.624	2.204	**2.301**
	5×5	2.855	2.991	3.131	12.200	**12.729**
	10×10	9.664	8.946	9.648	27.474	**33.158**
	20×20	8.801	9.431	8.736	38.430	**45.598**
	50×50	3.904	3.995	4.023	31.317	**36.581**
4096	2×2	0.641	0.615	0.701	**5.027**	4.976
	5×5	3.012	3.049	2.781	16.844	**21.606**
	10×10	9.181	9.558	9.610	37.649	**44.070**
	20×20	7.070	7.894	7.158	32.421	**37.425**
	50×50	3.944	4.025	4.025	**30.034**	25.448
8192	2×2	0.670	0.661	0.656	7.178	**9.727**
	5×5	3.101	3.114	3.244	37.084	**39.396**
	10×10	8.320	8.433	7.718	43.632	**51.513**
	20×20	6.311	6.487	6.642	30.501	**33.221**
16384	2×2	0.645	0.643	0.688	17.383	**17.892**
	5×5	3.229	3.273	3.131	49.275	**56.034**
	10×10	6.590	6.721	6.326	31.167	**35.912**
	20×20	5.992	6.253	6.100	28.455	**34.431**
32768	2×2	0.648	0.649	0.658	28.329	**29.571**
	5×5	2.635	2.661	2.426	48.156	**56.226**
	10×10	5.764	5.930	6.169	37.050	**41.890**
	20×20	5.611	5.961	5.596	34.616	**38.317**
65536	2×2	0.661	0.660	0.680	37.278	**41.438**
	5×5	2.199	2.237	2.156	39.669	**43.630**
	10×10	6.379	6.548	6.065	34.137	**37.650**

Таблица 12. Ускорение множественной свёртки OpenCL-программами на платформе NVidia

N	M×J	A	B	C	D	E
32	2×2	0.116	0.233	**0.400**	0.063	0.081
	5×5	0.787	1.308	**2.576**	0.429	0.462
	10×10	2.881	5.000	**8.095**	1.441	2.179
	20×20	9.414	17.063	**27.300**	5.151	6.825
	50×50	30.250	65.154	**77.000**	18.822	38.500
	100×100	32.962	65.923	**81.619**	30.607	53.563
	200×200	36.077	74.053	**93.800**	37.520	70.350
64	2×2	0.337	0.500	**0.814**	0.164	0.174
	5×5	1.802	2.646	**3.943**	0.954	1.030
	10×10	6.281	11.167	**16.080**	3.865	4.517
	20×20	17.694	35.389	**37.471**	9.800	14.814
	50×50	34.965	71.179	**79.720**	28.884	51.103
	100×100	40.192	75.075	**88.422**	38.260	71.054
	200×200	42.344	77.631	**98.803**	44.361	80.506
128	2×2	0.539	0.710	**1.000**	0.277	0.315
	5×5	3.640	4.440	**7.114**	1.932	2.484
	10×10	15.920	26.534	**31.841**	13.267	15.920
	20×20	31.400	52.333	**52.333**	22.429	39.250
	50×50	42.473	93.440	**116.800**	46.720	66.743
	100×100	48.575	80.958	**107.945**	51.132	75.309
	200×200	49.209	98.419	**121.131**	52.490	104.980
256	2×2	0.727	0.941	**1.311**	0.640	0.800
	5×5	4.955	6.812	**8.385**	4.192	5.450

	10×10	11.526	19.909	**24.333**	14.600	19.909
	20×20	31.545	57.833	**69.400**	38.556	49.571
	50×50	54.953	104.410	**116.011**	49.719	80.315
	100×100	52.125	94.773	**115.833**	54.868	99.286
	200×200	52.175	99.606	**117.393**	57.667	86.500
512	2×2	1.206	1.708	**1.864**	1.367	1.464
	5×5	6.438	8.822	**9.528**	5.955	8.822
	10×10	16.857	24.842	27.765	24.842	**39.333**
	20×20	36.950	67.182	67.182	43.471	**73.900**
	50×50	64.714	104.104	126.021	63.011	**126.021**
	100×100	60.089	107.887	**131.861**	65.027	115.781
	200×200	58.099	104.742	**127.998**	62.493	114.411
1024	2×2	1.429	1.818	1.818	2.963	**3.478**
	5×5	6.716	8.569	8.283	14.200	**17.138**
	10×10	18.811	26.237	27.694	35.607	**55.389**
	20×20	44.194	72.318	75.762	54.862	**93.588**
	50×50	66.867	132.063	**135.449**	74.402	124.294
	100×100	64.929	106.904	**139.257**	65.736	125.994
	200×200	61.224	112.536	**131.292**	64.772	124.382
2048	2×2	1.526	1.723	1.513	4.807	**6.214**
	5×5	7.287	8.132	8.502	28.774	**38.697**
	10×10	24.404	31.888	32.319	56.943	**95.664**
	20×20	47.576	80.052	78.349	68.193	**122.747**
	50×50	77.568	139.437	148.262	85.494	**162.676**
	100×100	69.470	119.994	**140.243**	76.064	132.775
4096	2×2	1.499	1.578	1.596	10.455	**13.069**
	5×5	7.874	8.574	8.825	40.300	**58.976**
	10×10	23.853	29.346	30.263	64.561	**107.602**
	20×20	56.910	83.549	82.670	83.549	**145.437**
	50×50	82.800	137.850	146.565	84.556	**147.751**
	100×100	75.480	129.630	**149.873**	83.597	143.826
8192	2×2	1.454	1.490	1.480	23.472	**25.606**
	5×5	8.210	8.794	8.929	55.851	**80.769**
	10×10	26.761	31.208	31.300	79.674	**131.463**
	20×20	59.489	86.613	84.426	90.359	**163.886**
	50×50	84.016	151.045	154.819	93.440	**174.717**
16384	2×2	1.499	1.517	1.530	28.231	**36.700**
	5×5	8.515	9.042	9.196	68.284	**106.575**
	10×10	26.673	31.106	31.495	90.944	**158.357**
	20×20	77.436	111.783	111.149	121.357	**210.856**
	50×50	100.41	174.666	---	114.121	**206.627**
32768	2×2	1.395	1.416	1.781	44.668	**60.578**
	5×5	8.121	8.488	10.298	78.675	**143.988**
	10×10	24.688	27.812	31.106	105.532	**185.465**
	20×20	65.389	90.039	65.087	113.569	**201.975**
65536	2×2	1.168	1.175	1.205	54.032	**98.826**
	5×5	6.290	6.455	6.635	105.052	**173.778**
	10×10	21.880	23.852	23.705	113.889	**201.758**
	20×20	33.502	38.298	---	131.304	**229.012**

Анализируя эти таблицы, можно заметить, что при N≥512 вариант **E** даёт наилучшие показатели ускорения (как на платформе «Intel», так и на платформе «NVidia») почти на всех наборах параметров (M,J,N). Но при N<512 (и на платформе «Intel», и на платформе «NVidia») для многих наборов параметров вариант E всё же уступает варианту C как наиболее оптимальному. И объясняется это тем, что процедуры ядра варианта C (как и в вариантах A,B) эффективно используют быстродействующую локальную (private) память исполнителей OpenCL-среды для размещения в ней обрабатываемых векторов.

Поскольку при каждом вызове процедуры ядра вариантов D и E совершают лишь одноразовые действия над элементами векторов, расположенных в заданных матрицах в глобальной памяти, то никакой выгоды от сохранения элементов этих векторов в локальной памяти получить не представляется возможным.

Преимущество варианта C как раз и проявляется в тех случаях, когда объём такой локальной памяти позволяет хранить в ней полностью один (XP) или даже два (XP,YP) вектора, над которыми выполняется интенсивная обработка в процедурах ядра. Но это преимущество тут же исчезает, когда текущий обрабатываемый вектор (один YP или оба YP и XP) становится невозможно целиком разместить в этой локальной памяти при больших длинах (N) векторов.

Заметим, что это ограничение, связанное с максимально возможным объёмом локальной памяти исполнителей OpenCL-среды, на разных OpenCL-платформах проявляется при разных значениях N. Так, на платформе «Intel» такое ограничение возникает уже при N=512, а на платформе «NVidia» − лишь при N=8192.

Подобное ограничение приходится как-то учитывать при составлении процедур ядра. Для этого можно, например, предусмотреть компоновку разных вариантов процедур, используя директивы условной компиляции. В одном варианте (при допустимой длине N) обрабатываемые вектора XP, YP располагать в локальной памяти (как это и было продемонстрировано в рассмотренных ранее процедурах). А в другом варианте сразу настраивать указатели векторов XP, YP на те участки глобальной памяти, куда они впоследствии и будут записываться.

Чтобы учесть обозначенное ограничение и предусмотреть различные варианты компоновки OpenCL-программы на любой из рассмотренных платформ, а также обеспечить работоспособность полученной OpenCL-программы при любых значениях длин векторов N, в процедурах ядра вариантов A, B ,C объявления векторов XP и YP были скорректированы так:

```
#if NP<=MAX_NP
 complex XP[NP];
#else
__global complex *XP; XP=VZ;
#endif
#if NP*2<=MAX_NP
 complex YP[NP];
#else
__global complex *YP; YP=VY;
#endif
```

Величины **NP** и **MAX_NP** задают соответственно текущую и максимально возможную (для данной платформы) длину комплексных

векторов, которые процедуры ядра могут разместить в локальной памяти исполнителей. Они определяются как define-переменные при компиляции файла “MFSvertka.cl”, содержащего исходный текст процедур ядра на языке OpenCL. Поясним, как обеспечить требуемую настройку величин **NP** и **MAX_NP** без необходимости корректировать компилируемый файл “MFSvertka.cl” всякий раз для нового значения N длины обрабатываемых векторов, которое может задаваться, например, в качестве параметра командной строки при запуске OpenCL-программы.

Ранее (в [3,п.2.4]) уже было продемонстрировано, как можно в OpenCL-программе осуществить компиляцию процедур ядер, сосредоточенных в файле. Для этого надо предварительно прочитать содержимое этого текстового файла в одну или несколько строк. Затем подключить в OpenCL-контекст новый программный объект **prgrm** и откомпилировать его в универсальный внутренний код, который далее будет приниматься на исполнение драйвером OpenCL-устройства:

```
cl_program prgrm
prgrm=clCreateProgramWithSource
(cntxt,1,(char**)&cSrcStr,0,0);
clBuildProgram(prgrm,0,NULL,NULL,
 BPrgrmOpt,NULL,NULL);
```

При вызове функции **clBuildProgram** можно поставить (5-м параметром) непустой указатель на строку, в которой можно задать опции такой компиляции. А саму эту строку следует предварительно подготовить такой, чтобы в ней были сформированы необходимые определения для define-переменных NP и MAX_NP:

```
char *Opt="-D NP=%d -D MAX_NP=%d";
 sprintf(BPrgrmOpt,Opt,N,MAX_NP);
```

В итоге значение NP принимается равным длине обрабатываемых векторов, т.е. значению N, которое задаётся в самой OpenCL-программе явно или передаётся в качестве параметра командной строки при запуске программы. А значение MAX_NP задаётся в OpenCL-программе с помощью директивы #define в зависимости от того, для какой платформы она компонуется:

```
#if USE_OPENCL==1 /* платформа INTEL
*/
#define MAX_NP 512
#elif USE_OPENCL==2 /* пл. NVIDEA */
#define MAX_NP 16384
#else
#define MAX_NP 256
#endif
```

7. Заключение

Таким образом, технологию OpenCL можно успешно применять для повышения быстродействия таких вычислительных программ, в которых требуется выполнять над множеством векторов, сгруппированных в многомерные массивы данных, хотя и весьма сложные, но, по сути, однотипные вычислительные операции.

Однако для этого требуется построить такую программу, которая сумеет обеспечить в среде OpenCL исполнение этих однотипных операций над разными векторами путём организации одновременной совместной вычислительной работы необходимого количества параллельно функционирующих исполнителей.

Представленные в статье примеры OpenCL-программ показывают, что операцию свёртки для многомерных массивов комплексных векторов в среде OpenCL можно ускорить даже в большей степени, чем множественную операцию быстрого преобразования Фурье для массивов комплексных векторов такого же размера.

Публикация выполнена в рамках государственного задания НИЦ «Курчатовский институт» — НИИСИ по теме «Методы разработки аппаратно-программных платформ на основе защищенных и устойчивых к сбоям систем на кристалле и сопроцессоров искусственного интеллекта и обработки сигналов», шифр № FNEF-2024-0003.

Acceleration of Fast Convolution Operation for Multiple Complex Vectors Based on OpenCL Technology

A.A. Burtsev

Abstract. The article is devoted to the use of OpenCL technology, which allows using the powerful resources of graphic processors to improve the performance of computing programs. Techniques for developing effective parallel programs in the OpenCL environment are considered to speed up the convolution operation for multiple complex vectors based on the fast Fourier transform.

Keywords: parallel programming, OpenCL technology, heterogeneous systems, fast Fourier transform, convolution operation.

Литература

1. В.В. Воеводин, В.В. Воеводин. Параллельные вычисления. Спб., БХВ-Петербург, 2004.

2. Официальный OpenCL–сайт организации Khronos Group, http://www.khronos.org/opencl/

3. А.А. Бурцев. Оптимизация операции перемножения матриц на основе технологии OpenCL. «Труды НИИСИ РАН», Т. 10 (2020), № 5-6, 100–112.

4. А.А. Бурцев. Применение технологии OpenCL для ускорения вычисления интегралов. // «Труды НИИСИ РАН», Т.13 (2023), №1-2, 19-24.

5. А.А. Бурцев. Ускорение быстрого преобразования Фурье на основе технологии OpenCL. «Труды НИИСИ РАН», Т. 11 (2021), № 4, 27–37.

6. А.А. Бурцев. Оптимизация операции быстрого преобразования Фурье в среде OpenCL. // «Труды НИИСИ РАН», Т.12 (2022), №1-2, 11-27.

7. А.А. Бурцев. Ускорение быстрого преобразования Фурье для многомерных массивов комплексных векторов на основе технологии OpenCL. // «Труды НИИСИ РАН», Т.14 (2024), №2, 22-30.

8. Стивен Смит. Цифровая обработка сигналов. Практическое руководство для инженеров и научных работников. М., Додека-XXI, 2012.