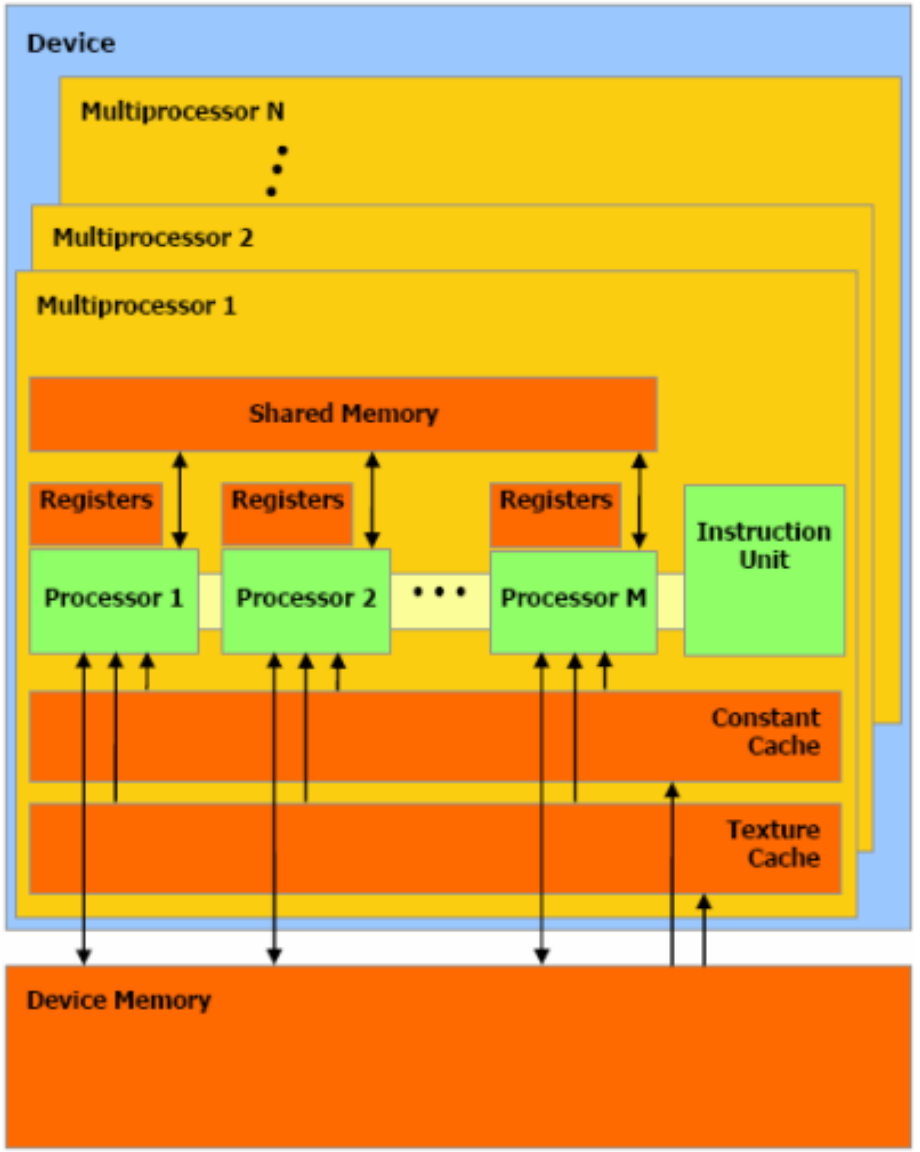
Использование расширяемых языков для программирования графических процессоров

Адинец А.В.[1,2]

[1]Объединённый институт ядерных исследований
[2]НИВЦ МГУ имени М.В. Ломоносова

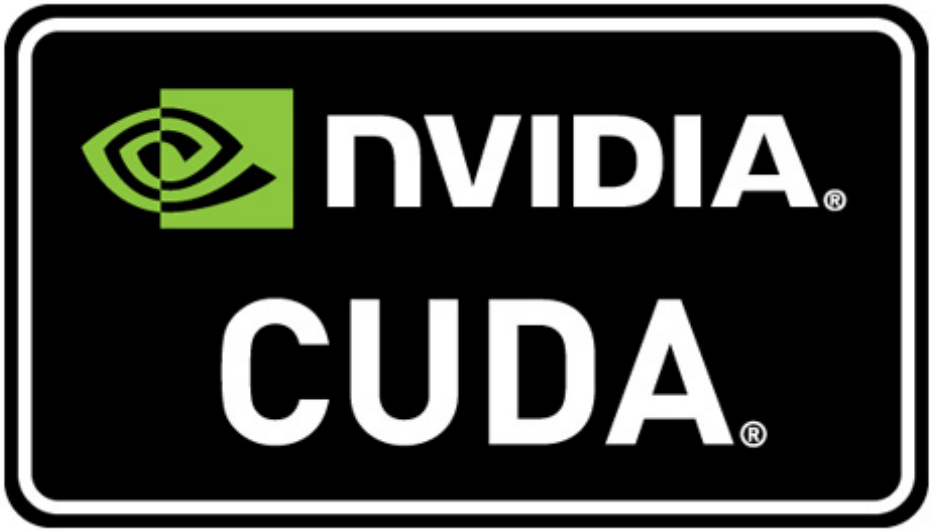
adinetz@gmail.com
twitter: @adinetz

CUDA

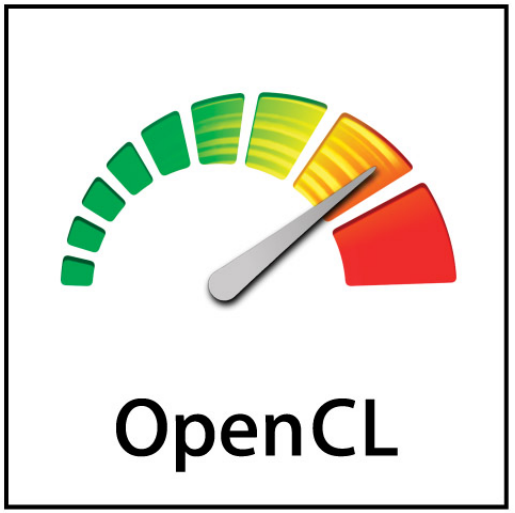


Низкоуровневый

Не везде



OpenCL



Везде

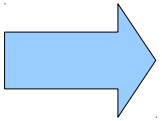
Низкоуровневый

Стандартный интерфейс

KHRONOS GROUP

Тонкая настройка

```
for(i = 0; i < n; i++)
  for(j = 0; j < n; j++) {
    double r = 0;
    for(k = 0; k < n; k++) {
      r += a[i * n + k] *
           b[k * n + j];
    }
    c[i * n + j] = r;
}
```



```
kernel void _N_nuwork__4917(global double* c_d, int c_l0, int
c_l1, global double* b_d, int b_l0, int b_l1, int n, global
double* a_d, int a_l0, int a_l1) {
  array2d_g_double_t c;c.d = c_d;
  c.l[0] = c_l0; c.l[1] = c_l1;
  array2d_g_double_t b;b.d = b_d;
  b.l[0] = b_l0; b.l[1] = b_l1;
  array2d_g_double_t a;a.d = a_d;
  a.l[0] = a_l0; a.l[1] = a_l1;
  int i_8 = get_global_id(0) * 8;
  int j_9 = get_global_id(1) * 2;
  double r_2_12 = 0; double r_2_15 = 0;
  double r_2_18 = 0; double r_2_21 = 0;
  double r_2_24 = 0; double r_2_27 = 0;
  double r_2_30 = 0; double r_2_33 = 0;
  double r_2_36 = 0; double r_2_39 = 0;
  double r_2_42 = 0; double r_2_45 = 0;
  double r_2_48 = 0; double r_2_51 = 0;
  double r_2_54 = 0; double r_2_57 = 0;
  for(int k = 0; k < n; ++k) {
    double aa_0_10 = a.d[a.l[1] * k + i_8];
    double aa_0_16 = a.d[a.l[1] * k + i_8 + 1];
    double aa_0_22 = a.d[a.l[1] * k + i_8 + 2];
    double aa_0_28 = a.d[a.l[1] * k + i_8 + 3];
    double aa_0_34 = a.d[a.l[1] * k + i_8 + 4];
    double aa_0_40 = a.d[a.l[1] * k + i_8 + 5];
    double aa_0_46 = a.d[a.l[1] * k + i_8 + 6];
    double aa_0_52 = a.d[a.l[1] * k + i_8 + 7];
    double bb_1_11 = b.d[b.l[1] * k + j_9];
    double bb_1_14 = b.d[b.l[1] * k + j_9 + 1];
    r_2_12 += aa_0_10 * bb_1_11; r_2_15 += aa_0_13 * bb_1_14;
    r_2_18 += aa_0_16 * bb_1_17; r_2_21 += aa_0_19 * bb_1_20;
    r_2_24 += aa_0_22 * bb_1_23; r_2_27 += aa_0_25 * bb_1_26;
    r_2_30 += aa_0_28 * bb_1_29; r_2_33 += aa_0_31 * bb_1_32;
    r_2_36 += aa_0_34 * bb_1_35; r_2_39 += aa_0_37 * bb_1_38;
    r_2_42 += aa_0_40 * bb_1_41; r_2_45 += aa_0_43 * bb_1_44;
    r_2_48 += aa_0_46 * bb_1_47; r_2_51 += aa_0_49 * bb_1_50;
    r_2_54 += aa_0_52 * bb_1_53; r_2_57 += aa_0_55 * bb_1_56;
  }
  c.d[c.l[1] * i_8 + j_9] = r_2_12;
  c.d[c.l[1] * i_8 + j_9 + 1] = r_2_15;
  c.d[c.l[1] * (i_8 + 1) + j_9] = r_2_18;
  c.d[c.l[1] * (i_8 + 1) + j_9 + 1] = r_2_21;
  c.d[c.l[1] * (i_8 + 2) + j_9] = r_2_24;
  c.d[c.l[1] * (i_8 + 2) + j_9 + 1] = r_2_27;
  c.d[c.l[1] * (i_8 + 3) + j_9] = r_2_30;
  c.d[c.l[1] * (i_8 + 3) + j_9 + 1] = r_2_33;
  c.d[c.l[1] * (i_8 + 4) + j_9] = r_2_36;
  c.d[c.l[1] * (i_8 + 4) + j_9 + 1] = r_2_39;
  c.d[c.l[1] * (i_8 + 5) + j_9] = r_2_42;
  c.d[c.l[1] * (i_8 + 5) + j_9 + 1] = r_2_45;
  c.d[c.l[1] * (i_8 + 6) + j_9] = r_2_48;
  c.d[c.l[1] * (i_8 + 6) + j_9 + 1] = r_2_51;
  c.d[c.l[1] * (i_8 + 7) + j_9] = r_2_54;
  c.d[c.l[1] * (i_8 + 7) + j_9 + 1] = r_2_57;
  }
```

Языковая поддержка

"Новый" язык

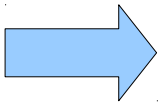
Дополнение к существующему

Расширяемые языки

Макросы Nemerle

```
macro nforMacro(header, body)
syntax ("nfor", "(", header, ")", body)
{
  // ...
}
```

```
nfor((i, j) in (m, n))
{
  // ...
}
```



```
for(i = 0; i < m; i++)
  for(j = 0; j < n; j++)
  {
    // ...
  }
```

Функции на ускорителе

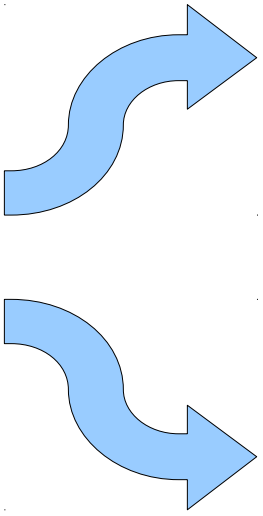
```
nukernel ker(b : nuarray1d[float], a :
nuarray1d[float]) : void { … }
```
- ядро

```
nucode someFun(i : int, j : int) : int
{ … }
```
- просто функция

```
nucall (0, [n], [16]) ker(b, a)
```
— вызов ядра

Цикл на ускорителе

```
nuwork(64) nfor(i in n) {
  a[i] = b[i] + c[i];
}
```



```
nukernel xxx_42(
  a : nuarray1d[float],
  b : nuarray1d[float],
  c : nuarray1d[float]
) {
  i = globalId(0);
  a[i] = b[i] + c[i];
}
```

```
nucall(currentIndex,
[n], [64])
xxx_42(a, b, c);
```

Обычное сложение массивов

```
Main() : void {

  def n = 1001;
  def a = array(n) : array[float];
  def b = array(n) : array[float];
  def c = array(n) : array[float];

  nfor(i in n) {a[i] = i; b[i] = i;}

  nfor(i in n) c[i] = a[i] + b[i];

  nfor(i in n) WriteLine(c[i]);

}
```

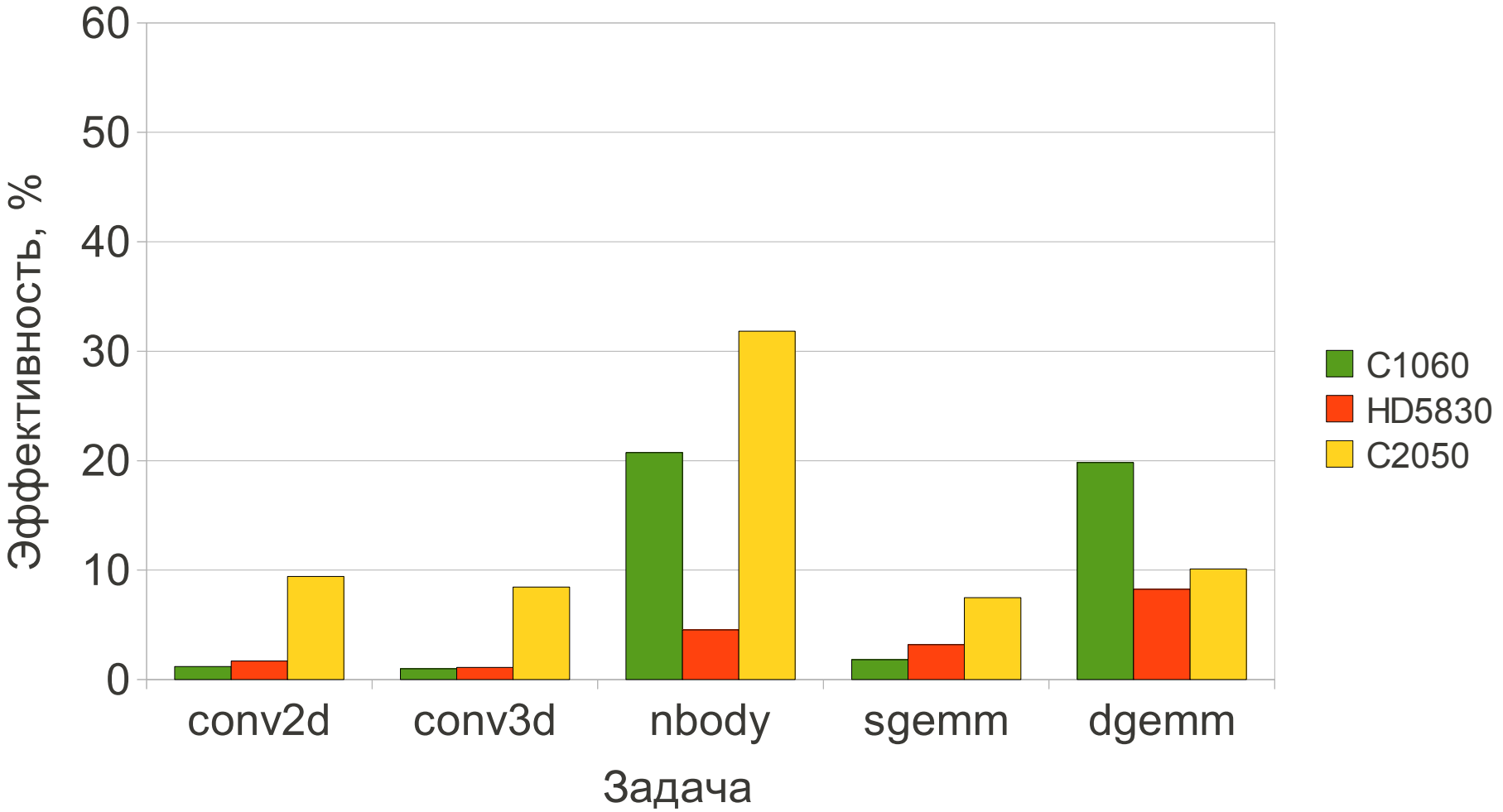
Сложение массивов на ГПУ

```
Main() : void {

  def n = 1001;
  def a = nunew array(n) : array[float];
  def b = nunew array(n) : array[float];
  def c = nunew array(n) : array[float];

  nfor(i in n) {a[i] = i; b[i] = i;}

  nuwork(64) nfor(i in n) c[i] = a[i] + b[i];

  nfor(i in n) WriteLine(c[i]);

}
```

Модельные задачи

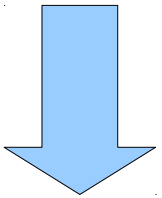
- 2D-свёртка 3x3 (conv2d)
- 3D-свёртка 3x3 (conv3d)
- задача N тел (nbody)
- умножение матриц, float (sgemm)
- умножение матриц, double (dgemm)

Простая реализация



Полная развёртка цикла

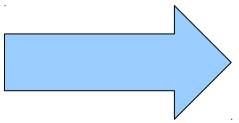
```
inline nfor((p, q) in (2, 2)){
  r += k[p, q] * a[i + p, j + q];
}
```



```
r += k[0, 0] * a[i, j];
r += k[0, 1] * a[i, j + 1];
r += k[0, 2] * a[i, j + 2];
r += k[1, 0] * a[i + 1, j];
r += k[1, 1] * a[i + 1, j + 1];
r += k[1, 2] * a[i + 1, j + 2];
r += k[2, 0] * a[i + 2, j];
r += k[2, 1] * a[i + 2, j + 1];
r += k[2, 2] * a[i + 2, j + 2];
```

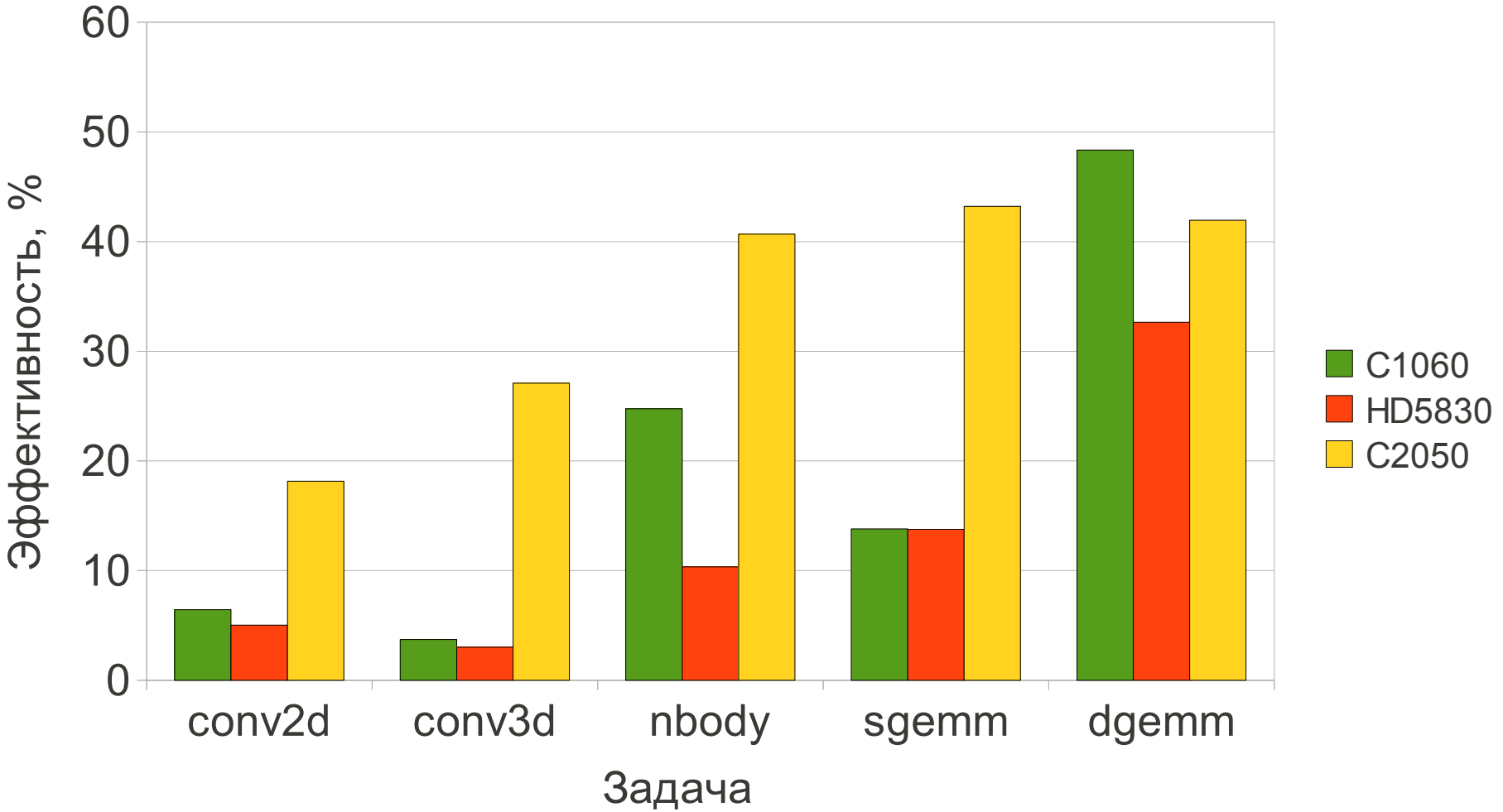
Глубокая развёртка

```
dmine(2) nfor(i in n) {
   mutable r = 0.0f;
   def aa = a[i];
  nfor(j in n) {
    def bb = b[j];
    r += f(aa, bb);
  }
  c[i] = r;
}
```

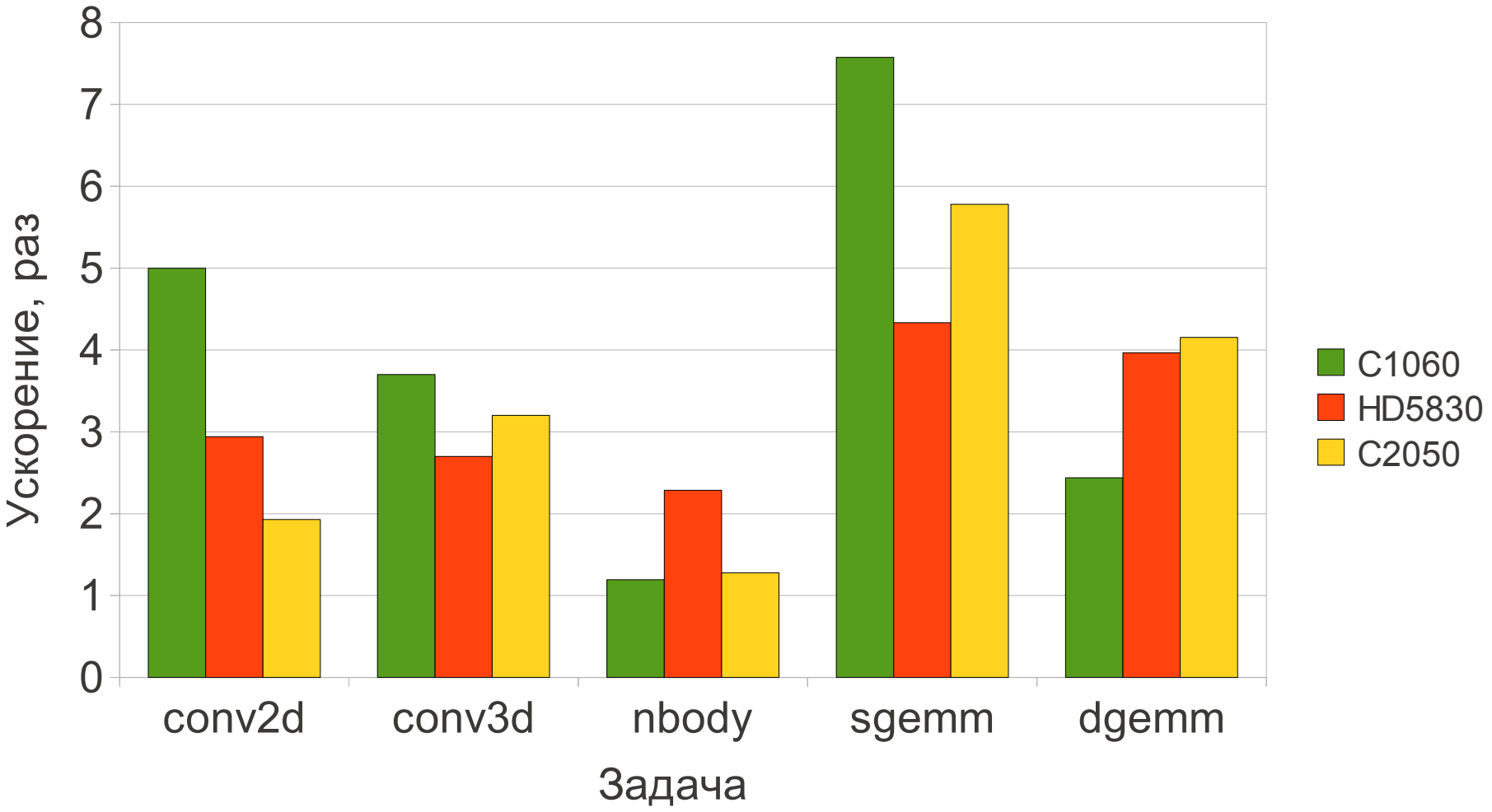


```
nfor(i1 in n :/ 2) {
  mutable r1 = 0.0f;
  mutable r2 = 0.0f;
  def aa1 = aa[i1];
  def aa2 = aa[i1 + 1];
  nfor(j in n) {
    def bb1 = b[j];
    def bb2 = b[j];
    r1 += f(aa1, bb1);
    r2 += f(aa2, bb2);
  }
  c[i1] = r1;
  c[i1 + 1] = r2;
}
nfor(i in 2 * (n / 2) <> n - 1)
{
  mutable r = 0.0f;
  def aa = a[i];
  nfor(j in n) {
    def bb = b[j];
    r += f(aa, bb);
  }
  c[i] = r;
}
```

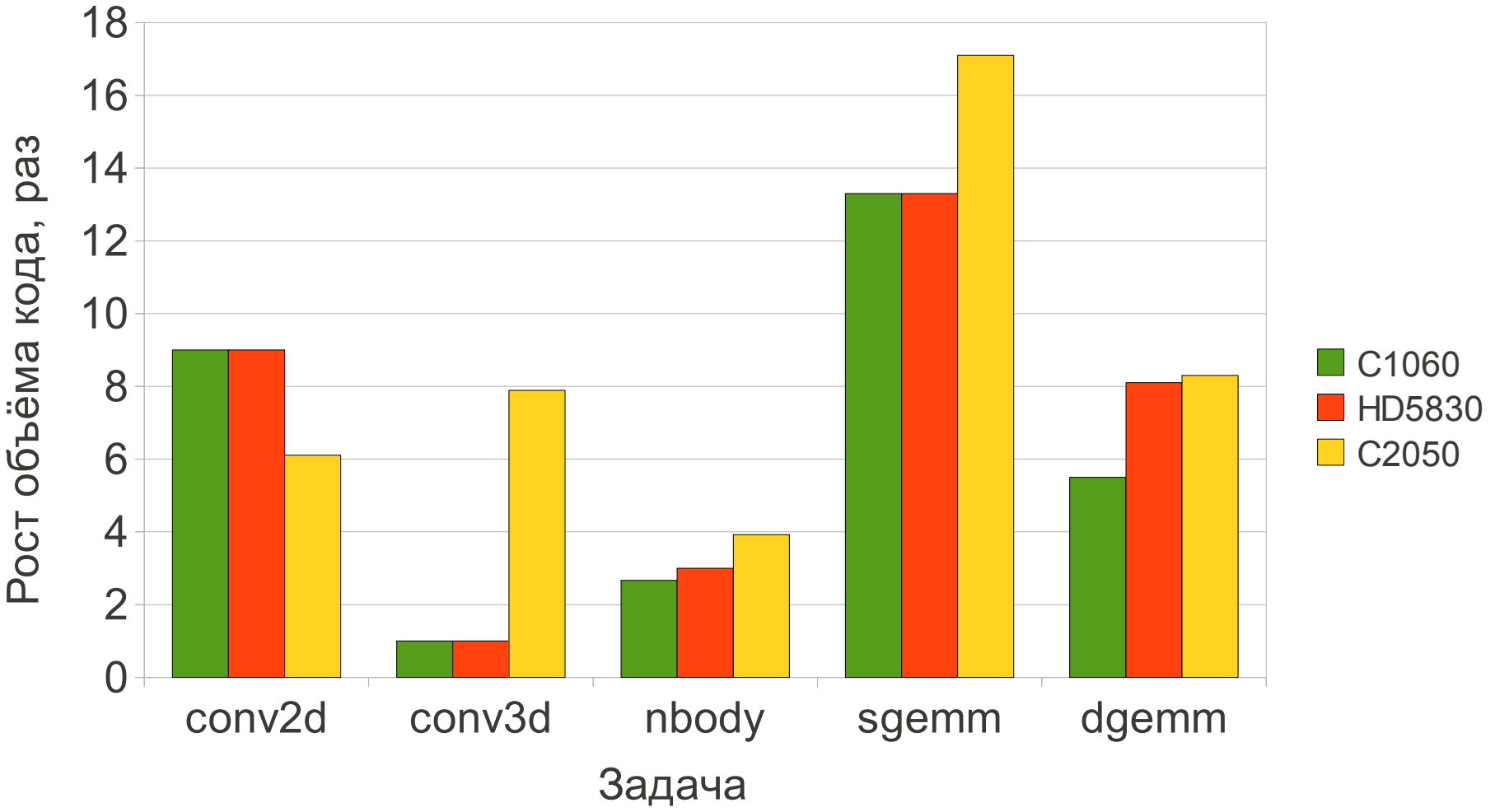
Эффективность с аннотациями



Ускорение

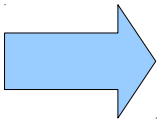

Ускорение, раз
0
1
2
3
4
5
6
7
8
conv2d
conv3d
nbody
sgemm
dgemm
Задача
C1060
HD5830
C2050

Рост объёма сгенерированного кода



Большой объём кода?

```
nuwork(2, 256) dmine(8, 2)
nfor((i, j) in (n, n)) {
  mutable r = 0.0f;
  nfor(j in n) {
    def aa = a[k, i];
    def bb = b[k, j];
    r += aa * bb;
  }
  c[i, j] = r;
}
```



```
kernel void _N_nuwork__4917(global double* c_d, int c_l0, int c_l1, global
double* b_d, int b_l0, int b_l1, int n, global double* a_d, int a_l0, int
a_l1) {
  array2d_g_double_t c;c.d = c_d;
  c.l[0] = c_l0; c.l[1] = c_l1;
  array2d_g_double_t b;b.d = b_d;
  b.l[0] = b_l0; b.l[1] = b_l1;
  array2d_g_double_t a;a.d = a_d;
  a.l[0] = a_l0; a.l[1] = a_l1;
  int i_8 = get_global_id(0) * 8;
  int j_9 = get_global_id(1) * 2;
  double r_2_12 = 0; double r_2_15 = 0;
  double r_2_18 = 0; double r_2_21 = 0;
  double r_2_24 = 0; double r_2_27 = 0;
  double r_2_30 = 0; double r_2_33 = 0;
  double r_2_36 = 0; double r_2_39 = 0;
  double r_2_42 = 0; double r_2_45 = 0;
  double r_2_48 = 0; double r_2_51 = 0;
  double r_2_54 = 0; double r_2_57 = 0;
  for(int k = 0; k < n; ++k) {
    double aa_0_10 = a.d[a.l[1] * k + i_8 + 1];
    double aa_0_16 = a.d[a.l[1] * k + i_8 + 2];
    double aa_0_22 = a.d[a.l[1] * k + i_8 + 3];
    double aa_0_28 = a.d[a.l[1] * k + i_8 + 4];
    double aa_0_34 = a.d[a.l[1] * k + i_8 + 5];
    double aa_0_40 = a.d[a.l[1] * k + i_8 + 6];
    double aa_0_46 = a.d[a.l[1] * k + i_8 + 7];
    double aa_0_52 = a.d[a.l[1] * k + i_8 + 8];
    double bb_1_11 = b.d[b.l[1] * k + j_9];
    double bb_1_14 = b.d[b.l[1] * k + j_9 + 1];
    r_2_12 += aa_0_10 * bb_1_11; r_2_15 += aa_0_13 * bb_1_14;
    r_2_18 += aa_0_16 * bb_1_17; r_2_21 += aa_0_19 * bb_1_20;
    r_2_24 += aa_0_22 * bb_1_23; r_2_27 += aa_0_25 * bb_1_26;
    r_2_30 += aa_0_28 * bb_1_29; r_2_33 += aa_0_31 * bb_1_32;
    r_2_36 += aa_0_34 * bb_1_35; r_2_39 += aa_0_37 * bb_1_38;
    r_2_42 += aa_0_40 * bb_1_41; r_2_45 += aa_0_43 * bb_1_44;
    r_2_48 += aa_0_46 * bb_1_47; r_2_51 += aa_0_49 * bb_1_50;
    r_2_54 += aa_0_52 * bb_1_53; r_2_57 += aa_0_55 * bb_1_56;
  }
  c.d[c.l[1] * (i_8 + 0) + j_9] = r_2_12;
  c.d[c.l[1] * (i_8 + 0) + j_9 + 1] = r_2_15;
  c.d[c.l[1] * (i_8 + 1) + j_9] = r_2_18;
  c.d[c.l[1] * (i_8 + 1) + j_9 + 1] = r_2_21;
  c.d[c.l[1] * (i_8 + 2) + j_9] = r_2_24;
  c.d[c.l[1] * (i_8 + 2) + j_9 + 1] = r_2_27;
  c.d[c.l[1] * (i_8 + 3) + j_9] = r_2_30;
  c.d[c.l[1] * (i_8 + 3) + j_9 + 1] = r_2_33;
  c.d[c.l[1] * (i_8 + 4) + j_9] = r_2_36;
  c.d[c.l[1] * (i_8 + 4) + j_9 + 1] = r_2_39;
  c.d[c.l[1] * (i_8 + 5) + j_9] = r_2_42;
  c.d[c.l[1] * (i_8 + 5) + j_9 + 1] = r_2_45;
  c.d[c.l[1] * (i_8 + 6) + j_9] = r_2_48;
  c.d[c.l[1] * (i_8 + 6) + j_9 + 1] = r_2_51;
  c.d[c.l[1] * (i_8 + 7) + j_9] = r_2_54;
  c.d[c.l[1] * (i_8 + 7) + j_9 + 1] = r_2_57;
  }
```

НИАСИЛИЛ, МНОГАБУКФ!!!

NUDA

Nemerle Unified Device Architecture

~12000 строк кода

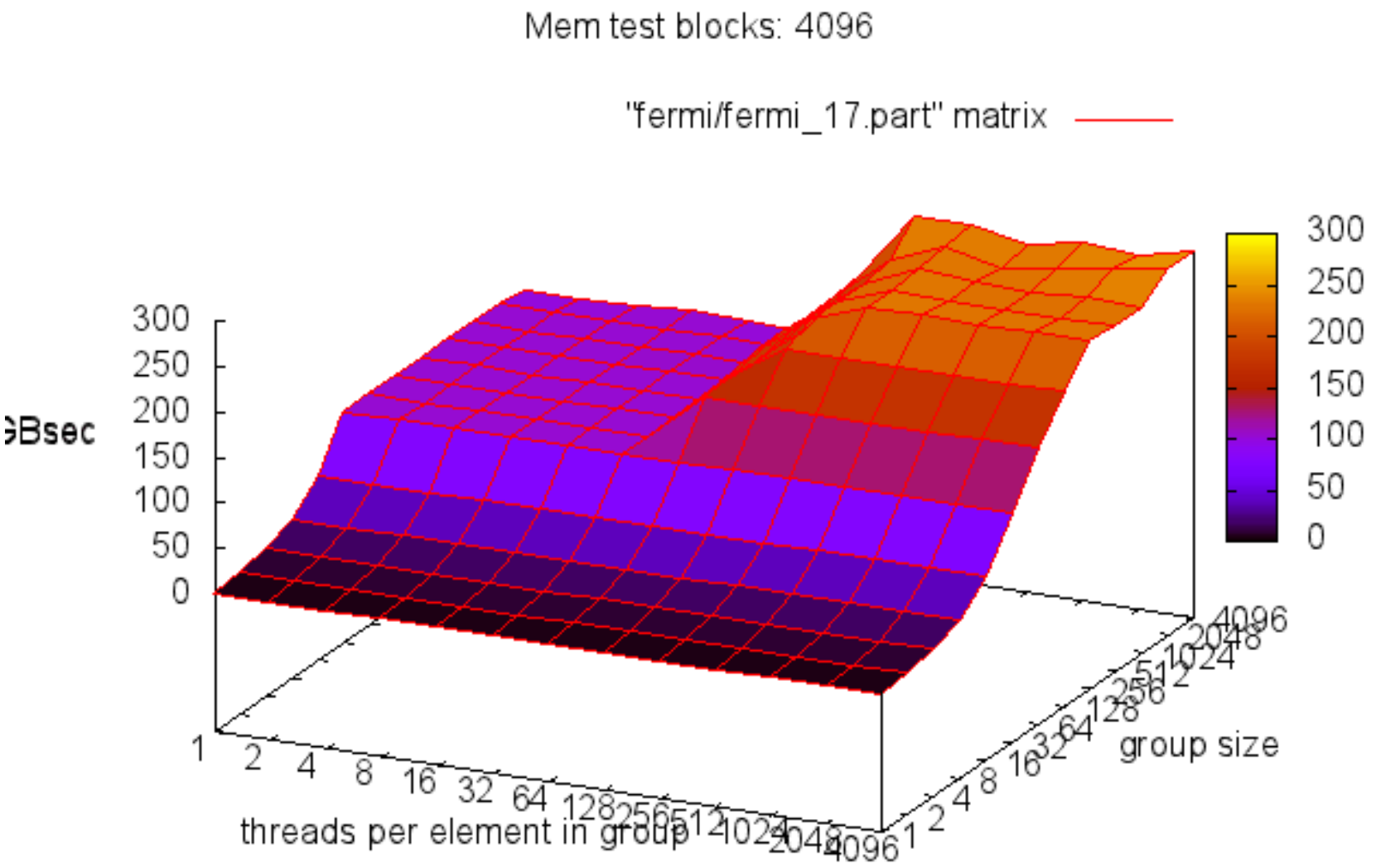
http://nuda.sf.net

Использование расширений

- Преобразования кода
  - Преобразования циклов, массивов
- Программирование ускорителей
- Оптимизации
- Параллельные конструкции
  - Асинхронные задачи
  - Атомарные операции
- Кластерные системы
  - Распределение данных
  - Удалённое исполнение

Вопросы?

Скорость чтения и когерентность

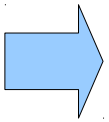


NVidia Tesla C2050 "Fermi"

Квазицитирование и интерполяция

- Подстановка выражений внутрь квазицитирования
  - $() — просто выражение
  - $(i:name) — выражение опр. типа
  - ..$() — список (сплайс)

```
def a : PExpr;
def b : list[PExpr];
def i : Name;
// …
def c = <[
  f($a, $(i:name));
  { ..$b }

]>;
```



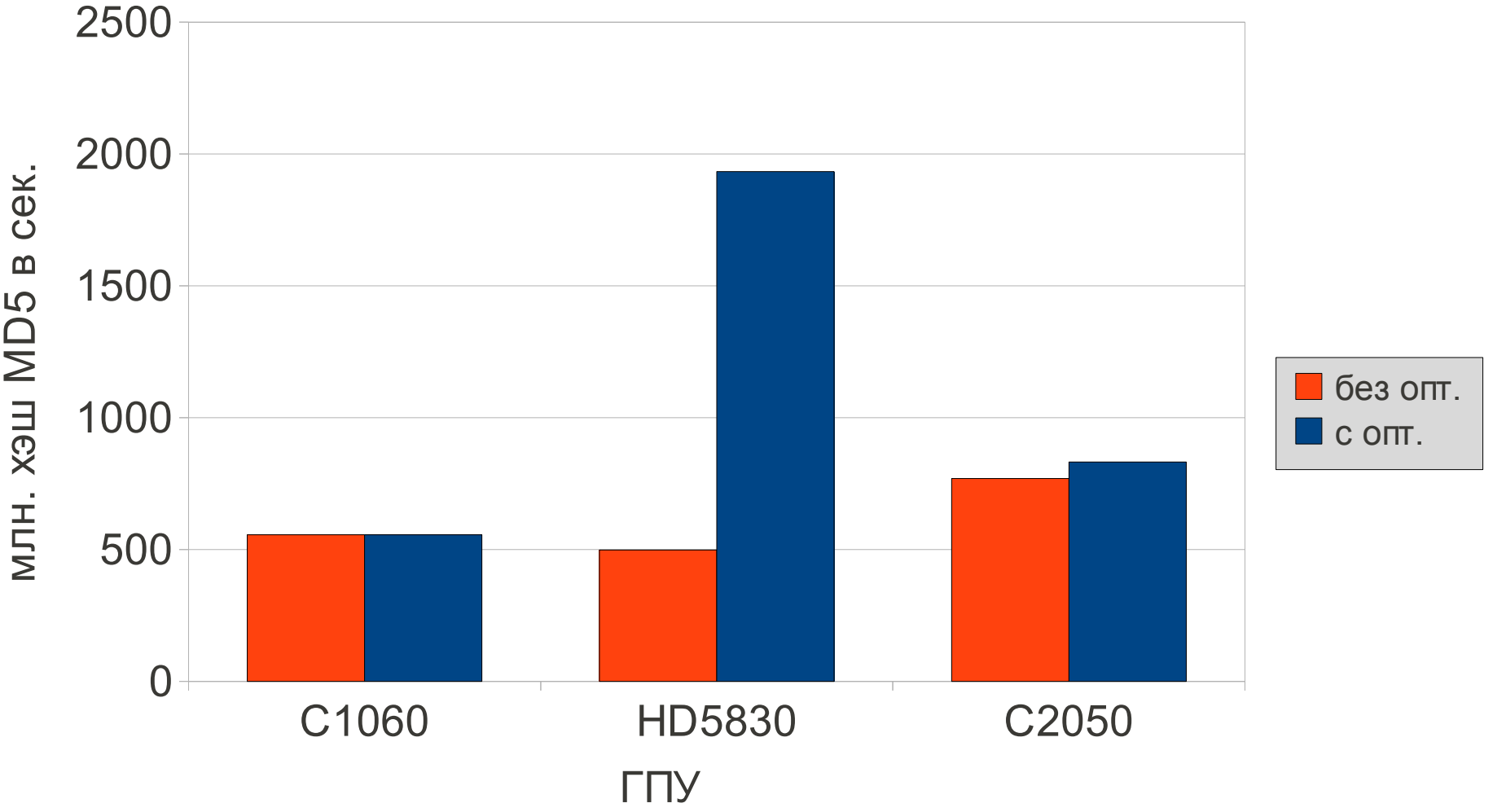
```
// ...
def c =
  PExpr.Sequence([
    PExpr.Call(PExpr.Ref(Name("f")), [
      a, PExpr.Ref(i)
    ]),
    PExpr.Sequence(b)
  ]);
```

Сопоставители

- Разбор фрагментов кода
- Более сложные преобразования

```
simpleEval(e : PExpr) : int {
  match(e) {
    | <[ $a + $b ]> => eval(a) + eval(b);
    | <[ $a - $b ]> => eval(a) – eval(b);
    | <[ $a * $b ]> => eval(a) * eval(b);
    | <[ $a / $b ]> => eval(a) / eval(b);
    | <[ $(i : int) ]> => i;
    | _ => WriteLine(@"can't evaluate"); -1;
  }
}
```

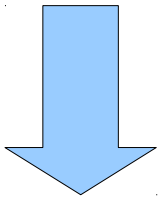
Производительность на MD5



Конфигурационные переменные

- Получают значения из командной строки

```
config("w") def nwalkers = 4096;
config def cutoff = 64;
config("V") def verbose = false;
config("a") def effNdevs = 1;
```



```
myprog -w20480 --cutoff=80 --eff-ndevs 1 --verbose
```

Атомарные операции

- Синтаксический сахар
  - **`atomic`** `{a[b[i]]++;}`
  - Локальные и глобальные

- Поддержка ограничена
  - Только самая внешняя операция
  - Только ++, --, +=, -=, &=, |=, ^=