Расширяемые языки и программирование высокопроизводительных вычислительных систем

Адинец А.В.[1,2]

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

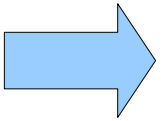
adinetz@gmail.com
twitter: @adinetz

Современные архитектуры

- Параллелизм, гетерогенность, иерархичность
- Уровни параллелизма
  - векторные команды — intrinsics, циклы
  - кэш 1, 2, 3 уровня — преобразования циклов
  - многоядерность — OpenMP
  - ускорители (ГПУ, Cell) — CUDA, OpenCL
  - многомашинность — MPI

Много кода

```
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;
  }
```

Модели программирования

- Новые модели программирования
  - Новые архитектуры
  - Просто возникают
- Для программиста
  - Продуктивность
  - Удобство
- Не только библиотеки
  - Расширения языков

Языки DARPA HPCS

- Языки:
  - Chapel, X10, Fortress
- Обещали в 2006 году:
  - "Увеличить производительность и продуктивность к 2010 году"
- Почему?
  - Бурный рост архитектур
  - Привязка к SMP-кластеру

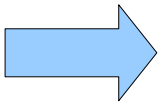
Как?

- Расширяемые языки
  - Ядро фиксировано
  - Синтаксис и семантика — расширяемы
- Возможности
  - Интеграция новых моделей
  - Удобство расширения
  - Плавный переход

Макросы

```
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++)
   {
      // ...
   }
```

NUDA

Nemerle Unified Device Architecture

~12000 строк кода

http://nuda.sf.net

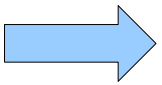
Nemerle: отличия от C++

- Нет указателей (`T*`)
  - Есть массивы (**`array`**`[T]`), списки (**`list`**`[T]`)
- Имя типа после переменной/функции
  - `f(x : float, n : int) : array[float]`
- Объявление переменных:
  - **`def`** `x = 5;` // нельзя присваивать
  - **`mutable`** `y : float;` // можно присваивать
- .NET-язык
  - использует платформу .NET

Цикл-гнездо nfor

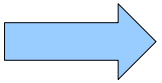
- Компактная форма записи тесно вложенного гнезда циклов

```
nfor((i, j) in (m, n)) {
  a[i, j] += b[i, j];
}
```



```
for(mutable i = 0; i < m; i++) {
  for(mutable j = 0; j < n; j++) {
    a[i, j] += b[i, j];
  }
}
```

```
nfor((i1, ..., iN) in
(d1, ..., dN))
  S;
```



```
for(i1 = a1; i1 <= b1; i1 += c1)
 ...
  for(iN = aN; iN <= bN; iN += cN)
    S;
```

```
d = a <> b :/ c

a <> b    <=>    a <> b :/ 1
n :/ c    <=>    0 <> n-1 :/ c
n         <=>    0 <> n-1 :/ 1
```

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
```

Данные на ускорителе

- Массивы для ускорителей:

```
def a = nunew array(n, n) : array[2, float];
```

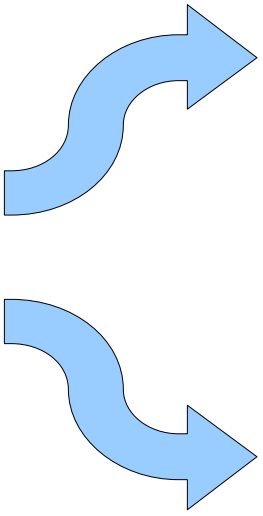
автоматическая синхронизация

- Классы памяти

```
def b = nulocal array(256) : array[float];
```

Цикл на ускорителе

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



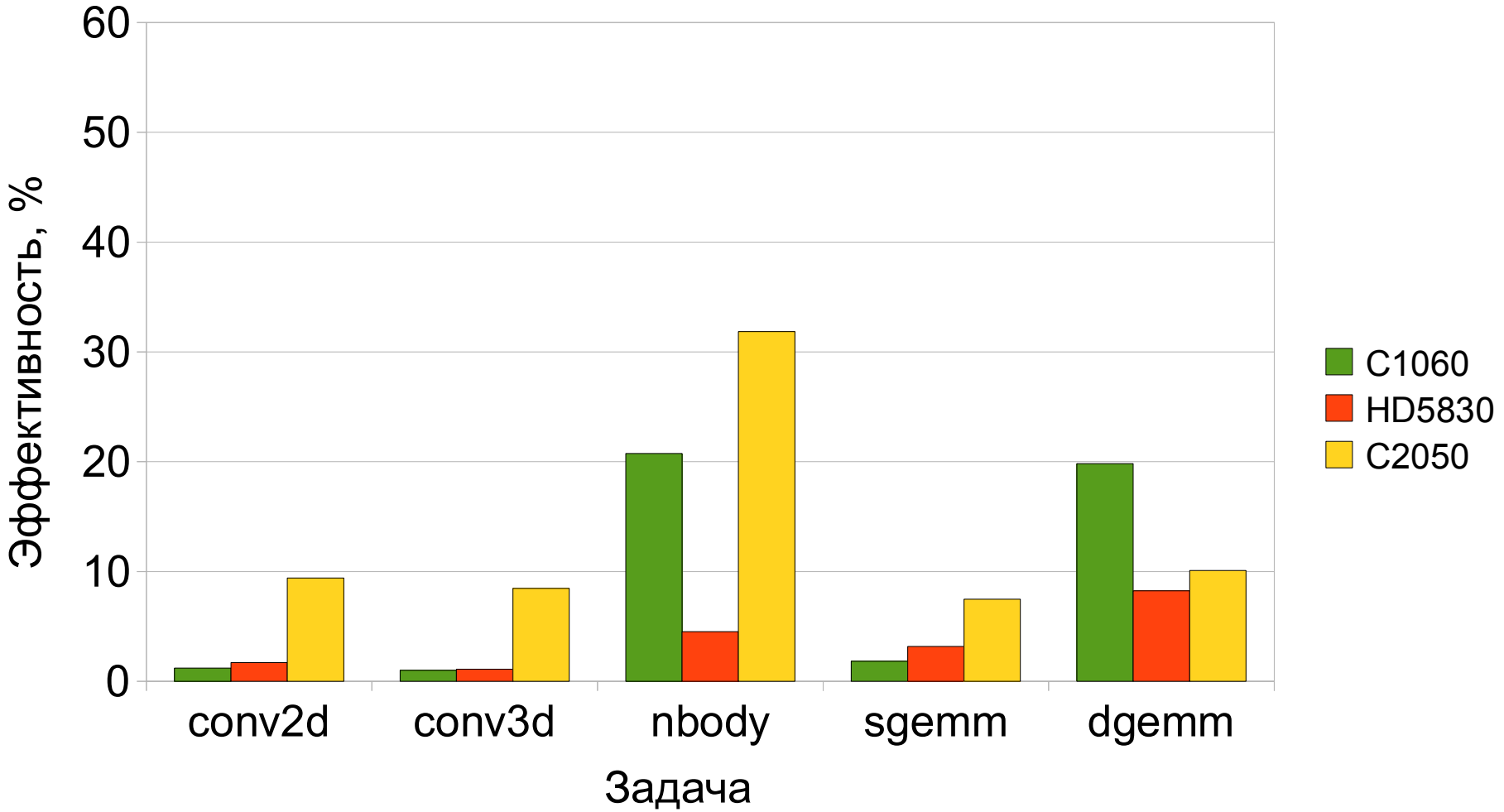
```
nukernel xxx_42(
  a : nuarray1d[float],
  b : nuarray1d[float],
  c : nuarray1d[float]
) {
  i = globalId(0);
  a[i] = b[i] + c[i];
}
```

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

Модельные задачи

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

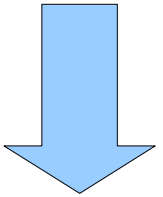
Простая реализация



Аннотация inline

- Полная развёртка цикла

```
inline nfor((p, q) in (-1 <> 1, -1 <> 1)){
  r += k[p + 1, q + 1] * a[i + p, j + q];
}
```

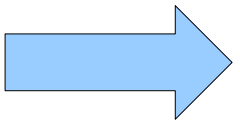


```
r += k[0, 0] * a[i – 1, j – 1];
r += k[0, 1] * a[i – 1, j];
r += k[0, 2] * a[i – 1, j + 1];
r += k[1, 0] * a[i, j – 1];
r += k[1, 1] * a[i, j];
r += k[1, 2] * a[i + 1, j + 1];
r += k[2, 0] * a[i, j – 1];
r += k[2, 1] * a[i, j];
r += k[2, 2] * a[i + 1, j + 1];
```

Аннотация dmine

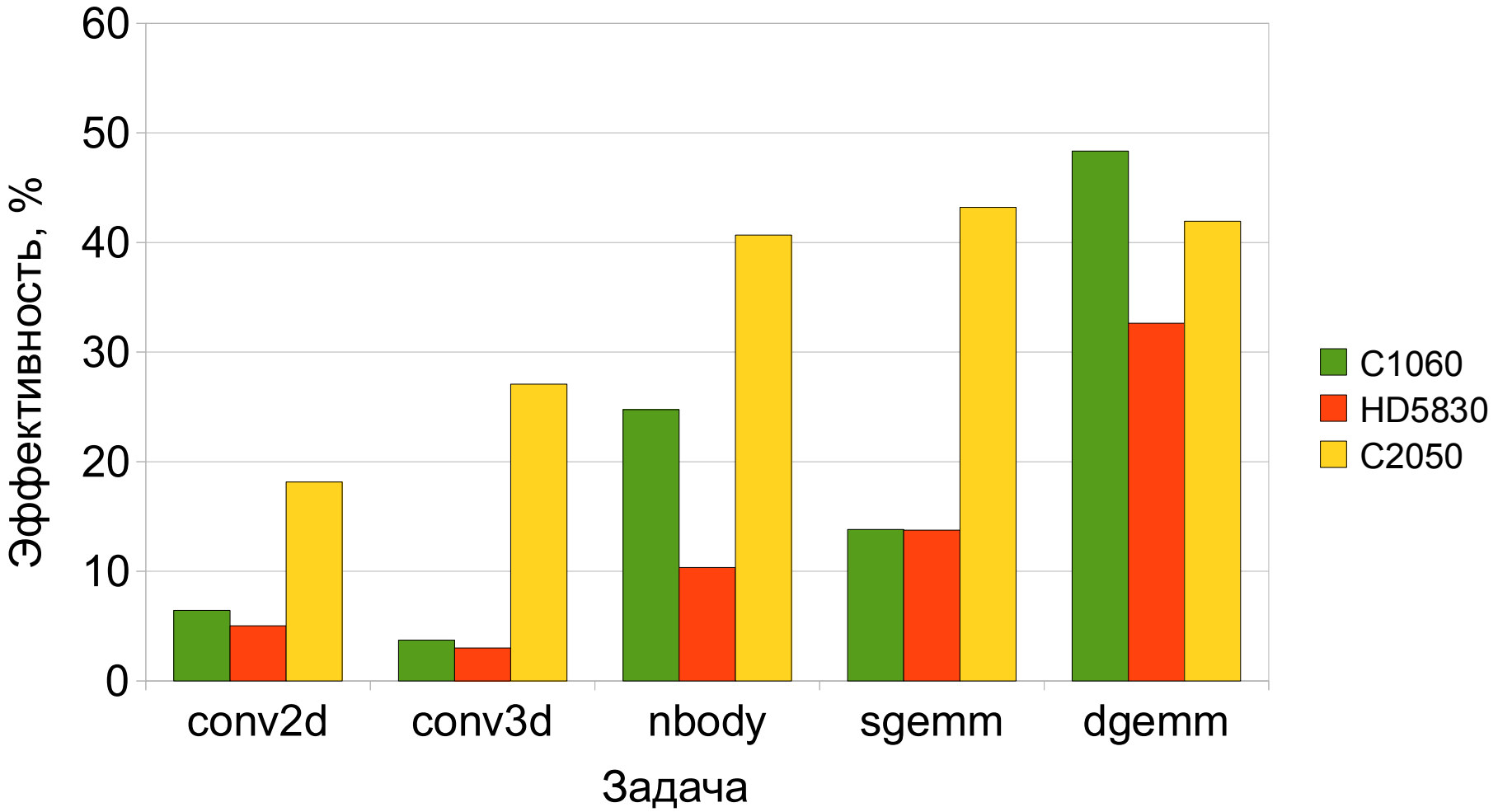
- Глубокая развёртка цикла

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

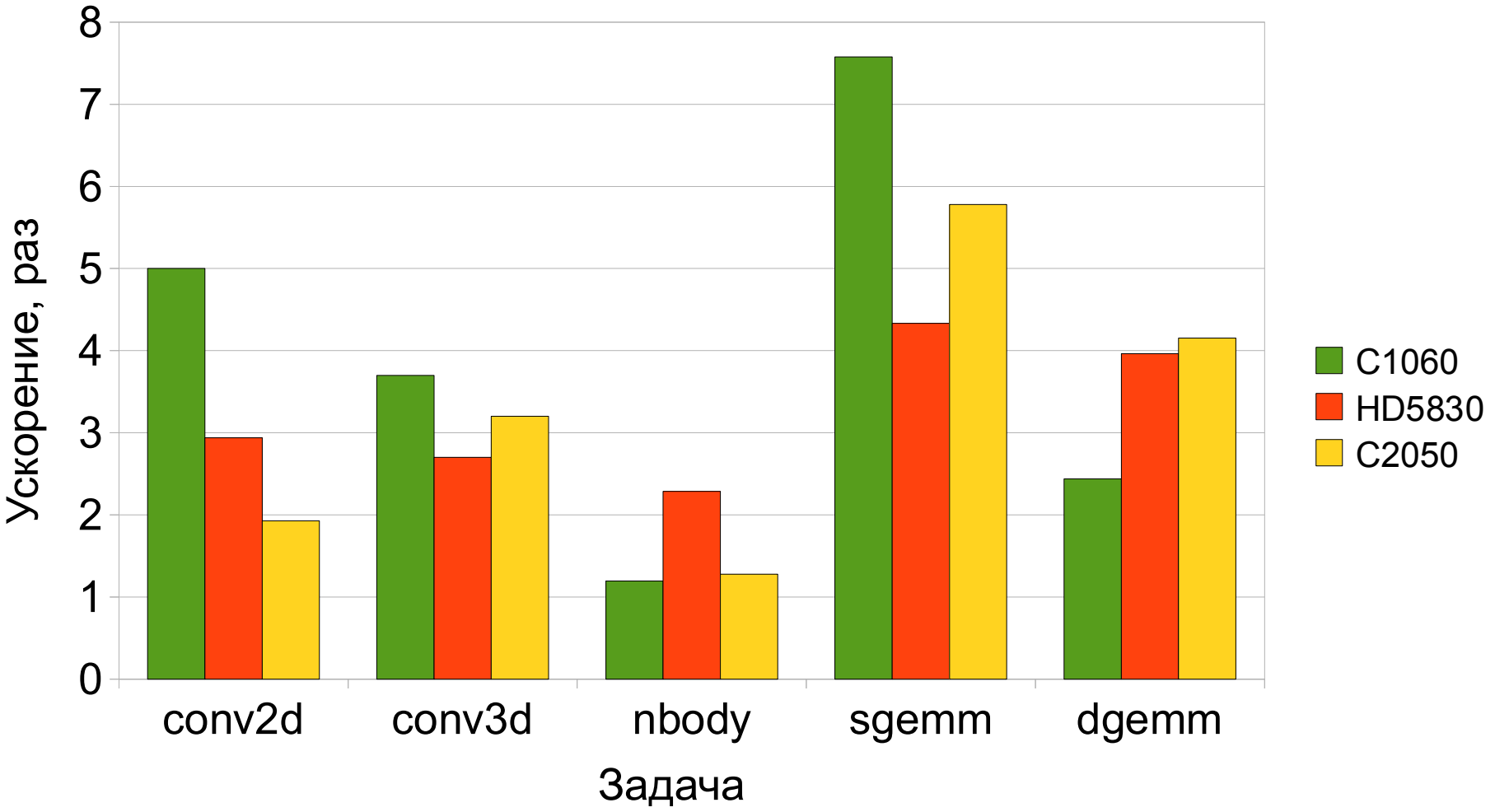


```
nfor(i1 in n :/ 2) {
  mutable r1 = 0.0f;
  mutable r2 = 0.0f;
  def aa1 = aa[i1];
  def aa2 = aa[i1 + 1];
  nfor(j in n) {
    def bb1 = b[j];
    def bb2 = b[j];
    r1 += aa1 * bb1;
    r2 += aa2 * bb2;
  }
  c[i1] = r1;
  c[i1 + 1] = r2;
}
nfor(i in 2 * n / 2 <> n - 1) {
  mutable r = 0.0f;
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

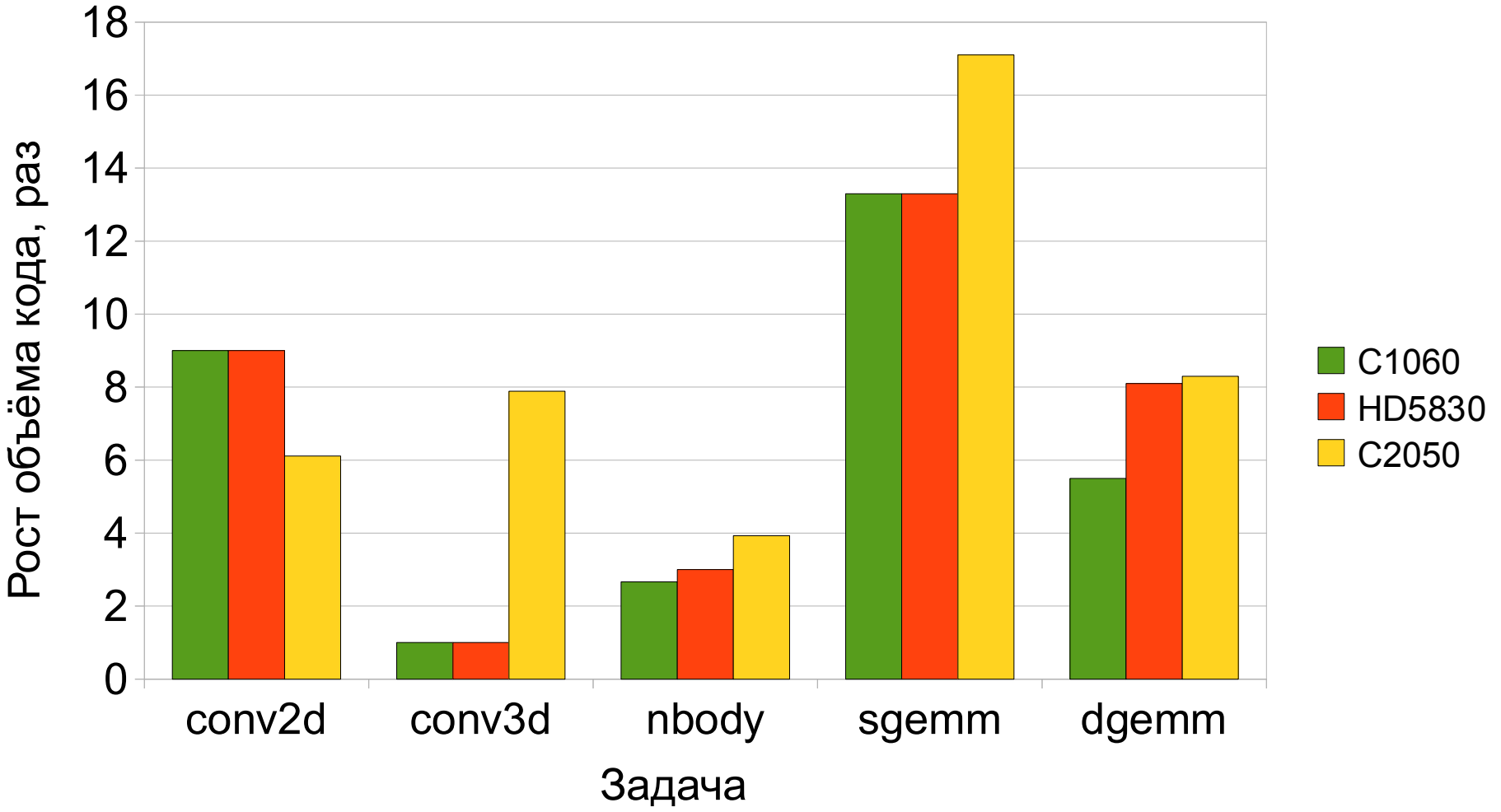
Эффективность с аннотациями



Ускорение

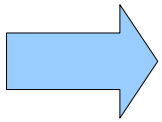

Ускорение, раз
0
1
2
3
4
5
6
7
8
conv2d
conv3d
nbody
sgemm
dgemm
Задача
C1060
HD5830
C2050

Рост объёма сгенерированного кода



Рост объёма 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;
}
```



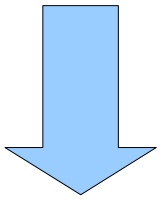
```
kernel void _N_nuwork__4917(global double* c_d, int c_l0, int c_l1, global
double* b_d, int b_l0, int b_l1, int n, global double* a_d, int a_l0, int
a_l1) {
  array2d_g_double_t c;c.d = c_d;
  c.l[0] = c_l0; c.l[1] = c_l1;
  array2d_g_double_t b;b.d = b_d;
  b.l[0] = b_l0; b.l[1] = b_l1;
  array2d_g_double_t 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 + 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 + 1];
    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!!!

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 max, min, ++, --, +, -, &, |, ^

Что такое расширяемый язык?

- Синтаксис и семантика изменяемы

- Любой язык :)
  - написать свой компилятор, потом расширить
  - расширить публичный компилятор

Что такое расширяемый язык? (2)

- Синтаксис и семантика изменяемы
- Для расширений
  - возможность заложена в спецификацию
  - независимость от компилятора
  - «мощный» язык
    - чаще всего — тот же самый
  - специальные конструкции языка

Языки как расширения

- Языки
  - GCC, CUDA, OpenCL, PGI Accelerator, CAPS HMPP, OpenMP, Cilk, T-Система, DVM, MC#, UPC, HPF, Titanium, X10
- Типы расширений
  - Спец. конструкции
  - Аннотации
  - Модификаторы переменных и функций
- Реализация
  - Библиотека + преобразование кода

Использование расширений

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

Стандартные макросы Nemerle

- "Стандартные" операторы
  - if .. else, for, foreach, while, do .. while, break, continue, return
- Параллельные конструкции
  - async, lock
  - "хорды" из MC#
  - кэширование значений функций

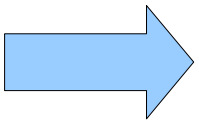
Функции расширяемого языка

- Расширение самого себя
- Преобразование программ
  - на самом себе
- Язык программирования
  - желательно хороший

Квазицитирование

- Запись деревьев кода при помощи самого кода

```
def someCode = <[
  def a = 1;
  def b = 2;
  def c = f(a, b);
]>;
```



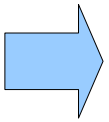
```
def someCode =
  PExpr.Sequence([
    PExpr.Define(PExpr.Ref(Name("a")),
      PExpr.Literal(Literal.Integer(1))),
    PExpr.Define(PExpr.Ref(Name("b")),
      PExpr.Literal(Literal.Integer(2))),
    PExpr.Define(PExpr.Ref(Name("c")),
      PExpr.Call(PExpr.Ref(Name("f")),
        [PExpr.Ref(Name("a")),
           PExpr.Ref(Name("b"))
 ]))]);
```

Интерполяция кода

- Подстановка выражений внутрь цитирования
  - $() — 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)
  ]);
```

Цитирование в LISP

- Простое
  - `'(list a b c)`
- С интерполяцией
  - `` `(list ,a ,b ,c) ``
- Сплайс
  - `` `(list ,a ,@list-of-args ,c) ``

Макрос

- Исполняется при компиляции
- Возвращает фрагмент кода
  - Подставляется на место вызова макроса
  - Вычисляется во время выполнения
- Если макросов несколько
  - От менее к более вложенному

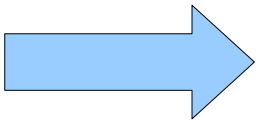
Макросы (Nemerle)

```
using System.Console;
// …
macro helloWorld(a, b) {
  WriteLine("Hello, World!");
  <[
    WriteLine("Hello, World!");
    $a + $b;
  ]>;
}
```

Во время компиляции:

```
Hello, World!
```

```
// ...
def c = helloWorld(1, 2);
// ...
```



В программу:

```
// ...
def c = {
  Console.WriteLine(
    "Hello, World");
  1 + 2;
};
// ...
```

Макросы (LISP)

- Определение
  - (**defmacro** <имя> (<параметры>) <тело>)
- Пример

```
(defmacro hello (a b)
  (format t "hello, world!")
  `((format t "hello, world!") (+ ,a ,b)))
```

- Расширение
  - (macroexpand-1 <выражение>)
  - (macroexpand-1 '(hello 1 2))

Задача: максимум

- Написать макрос, вычисляющий максимум двух выражений

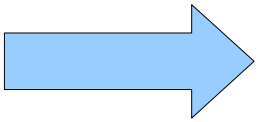
```
// ...
c = max(a, b);
// ...
```

```
// ...
c = if(a > b) a else b;
// ...
```

Побочный эффект

```
macro max(a, b) {
  <[ if($a > $b) $a else $b ]>
}
```

```
max({++a; a}, {++b; b});
```



```
if({++a; a} > {++b; b})
  {++a; a}
else
  {++b; b}
```

Захват переменной

```
macro max(a, b) {
  <[
    def va = $a;
    def vb = $b;
    if(va > vb) va else vb
  ]>
}
```

```
max(a, {va++; va});
```

```
def va = a;
def vb = {va++; va};
if(va > vb)
  va
else
  vb
```

Гигиена макросов

- Переменные в макросах не должны скрывать внешних переменных
- Common LISP: функция gensym
  - ```
    (defmacro maxmacro (a b)
      (let ((va (gensym)) (vb (gensym))
        `(let ((,va ,a) (,vb ,b)) (if (> ,va ,vb) ,va ,vb) ))))
    ```
- Nemerle: автоматически
  - Предыдущий пример работает!
    ```

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;
  }
}
```

Взаимодействие с компилятором

- Сообщения компиляции
  - `Message.{Hint, Warning, Error}()`
- Контекст
  - `ImplicitCTX()` внутри макроса
  - типизация `TypeExpr()`
- Параметры командной строки
  - для расширений — в Nemerle не поддерживаются

Синтаксические расширения

- Вызов макроса через синтаксис

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

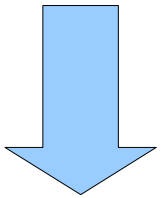
```
nfor(i in n) {
  a[i] = b[i];
}
```

```
nforMacro(
  i in n,
  {a[i] = b[i]}
);
```

- В Nemerle очень ограничен
  - только выражения (не списки и т.д.)
  - без вложенных правил (но есть Optional)
  - нет спецификации приоритетов

Задача: полная развёртка цикла

```
inline for(mutable i = 0; i < 3; i++) {
  a[i] = b[i];
}
```



```
a[0] = b[0];
a[1] = b[1];
a[2] = b[2];
```

Полная развёртка цикла

```
macro inlineMacro(aloop)
syntax("inline", aloop) {
  match(aloop) {
    | <[ for(mutable $(i : name) = $(a : int);
      $(i1 : name) < $(b : int);
      $(i2 : name)++) $body ]> =>
      if(i.Equals(i1) && i.Equals(i2)) {
        mutable resl = [] : list[PExpr];
        for(mutable vi = a; vi < b; vi++) {
          resl ::= <[ $(i : name) = $(vi : int); $body ]>;
        }
        resl = resl.Reverse();
        <[ { mutable $(i:name) : int; ..$resl; } ]>;
      } else {
        Message.Error(aloop.Location,
          "variable names do not match");
      }
    | _ =>
      Message.Error(aloop.Location, "loop expected");
  }
}
```

Стратегии

- Преобразования выражений
  - Можно обойтись функциями
- Стратегии более удобны
- `type strategy = PExpr -> PExpr`
- Возвращают:
  - в случае успеха — преобразованное выражение
  - в случае неудачи — `null`

Применение к выражению

- s — стратегия, e — выражение
- `s(e)`
  - а если стратегия — сложного вида?
  - если всегда нужен результат?
- `e !~ s` **<=>** `s(e)`
  - Обычное применение, внутри стратегий
- `e ~~ s` **<=>**

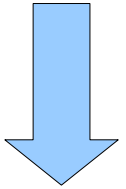
```
{def e1 = s(e); if(null != e1) e1 else e;}
```

  - Внутри макросов
- `e !~= s, e ~= s`

Правила

- Правило — простейшая стратегия:

```
<[ $(a : int) + $(b : int) ]> --> <[ $((a + b) : int) ]>
```



```
def arule(e : PExpr) : PExpr {
  match(e) {
    | <[ $(a : int) + $(b : int) ]> => <[ $((a + b) : int) ]>
    | _ => null;
  }
}
```

- Интересны прежде всего цепочки правил

Сложные стратегии

- Последовательное применение
  - s1 ~& s2 — обе должны быть успешны
  - s1 ~| s2 — хотя бы одна
  - try(s) — всегда успешный вариант s
  - srepeat(s1) — "до посинения" :)

Обход деревьев кода

- Разбор одной вершины
  - members(e), clone(e, es)
- В одной вершине
  - sall(s, atLeaf), sone(s, atLeaf), sany(s, atLeaf)
- Обход
  - bottomUp(s), topDown(s)

Пример: замена переменной

- Все вхождения переменной заменяются на выражение:

```
replaceVar(i : Name, er : PExpr) : strategy {
  def s(e : PExpr) : PExpr {
    match(e) {
      | <[ $(i1 : name) ]> =>
        if(i1.Equals(i)) er else null;
      | _ => null;
    }
  }
  bottomUp(s);
}
```

Пример: inline с заменой переменной

```
macro inlineMacro(aloop)
syntax("inline", aloop) {
  match(aloop) {
    | <[ for(mutable $(i : name) = $(a : int);
      $(i1 : name) < $(b : int);
      $(i2 : name)++) $body ]> =>
      if(i.Equals(i1) && i.Equals(i2)) {
        mutable resl = [] : list[PExpr];
        for(mutable vi = a; vi < b; vi++) {
          resl ::= body ~~ replaceVar(i, <[$(vi:int)]>);
        }
        resl = resl.Reverse();
        <[ { mutable $(i:name) : int; ..$resl; } ]>;
      } else {
        Message.Error(aloop.Location,
          "variable names do not match");
      }
    | _ =>
      Message.Error(aloop.Location, "loop expected");
  }
}
```

Язык Stratego

- DSL для
  - Описания синтаксиса
  - Преобразования программ
- Построен на стратегиях и правилах
- Поддержка динамических правил
- http://strategoxt.org/

Язык xoc

- Расширяемый вариант C
  - Расширения пишутся на языке Zeta
- Возможности расширения:
  - Наборы расширений
  - Цитирование, разбор
  - Спецификация грамматики
  - Атрибуты выражений
- http://pdos.csail.mit.edu/xoc/

Набор расширений

- C99 — базовый язык
- + — добавление расширений
  - C99+X+Y+Z — C99 с расширениями X, Y, Z
  - ? - произвольный набор расширений
  - коммутативна, ассоциативна
- Порядок расширений
- Используется при цитировании и разборе

Спецификация грамматики

- Расширение синтаксиса
  - Группы ассоциативности и приоритета
  - Обнаружение неоднозначности при разборе

```
grammar XRotate extends C99 {
  expr : expr "<<<" expr [Shift]
  expr : expr ">>>" expr [Shift]
}
```

Атрибуты

- Ленивое вычисление
- Синтаксис обращения к полю

- Обязательно переопределять:
  - Определение типа
  - Опускание (компиляцию)

XRotate: атрибут типа

```
extend attribute
type(term: ptr C.expr) : ptr Type {
  switch(term) {
    case ~expr{\a >>> \b} || ~expr{\a <<< \b}:
      if(a.type.isinteger && b.type.isinteger)
        return promoteunary(a.type);
      error(term.line, "non-integer rotate");
      return nil;
  }
  return default(term);
}
```

XRotate: атрибут опускания (компиляции)

```
extend attribute
compiled(term: ptr C.term) : ptr COutput.term {
  switch(term) {
    case ~{\a <<< \b}:
      n := a.type.sizeof * 8;
      return `C.expr{({
        \(a.type) x = \a;
        \(b.type) y = \b;
        (x << y) | (x >> (\n - y));
      })}.compiled;
    case ~{\a >>> \b}:
      n := a.type.sizeof * 8;
      return `C.expr{\a <<< (\n - \b)}.compiled;
  }
  return default(term);
}
```

Расширение семантики

- Массивное программирование
  - Продвижение функции/операции на массив
  - `a : array[int], b : array[int] => a + b`
- Методы/поля-расширения
  - Добавить в класс метод, которого там не было, не изменяя самого класса :)
  - Обращение — как к члену
- Конструкция `with() {}`
- Поиск функции с префиксом

Требования к создаваемому расширяемому языку

- Генерация эффективного машинного кода
  - Например, генерация в C
- Достаточная расширяемость и гибкость
  - Расширения синтаксиса и семантики
  - Неортогональные преобразования кода
  - Создание новых моделей программирования
- Интеграция с существующими системами

Вопросы и проблемы

- С чего начать?
- Как расширять семантику?
- Взаимодействие расширений
- Обработка ошибок
- Преобразования:
  - Строгая проверка
  - Всё дозволено
  - Что-то посередине

Вопросы?

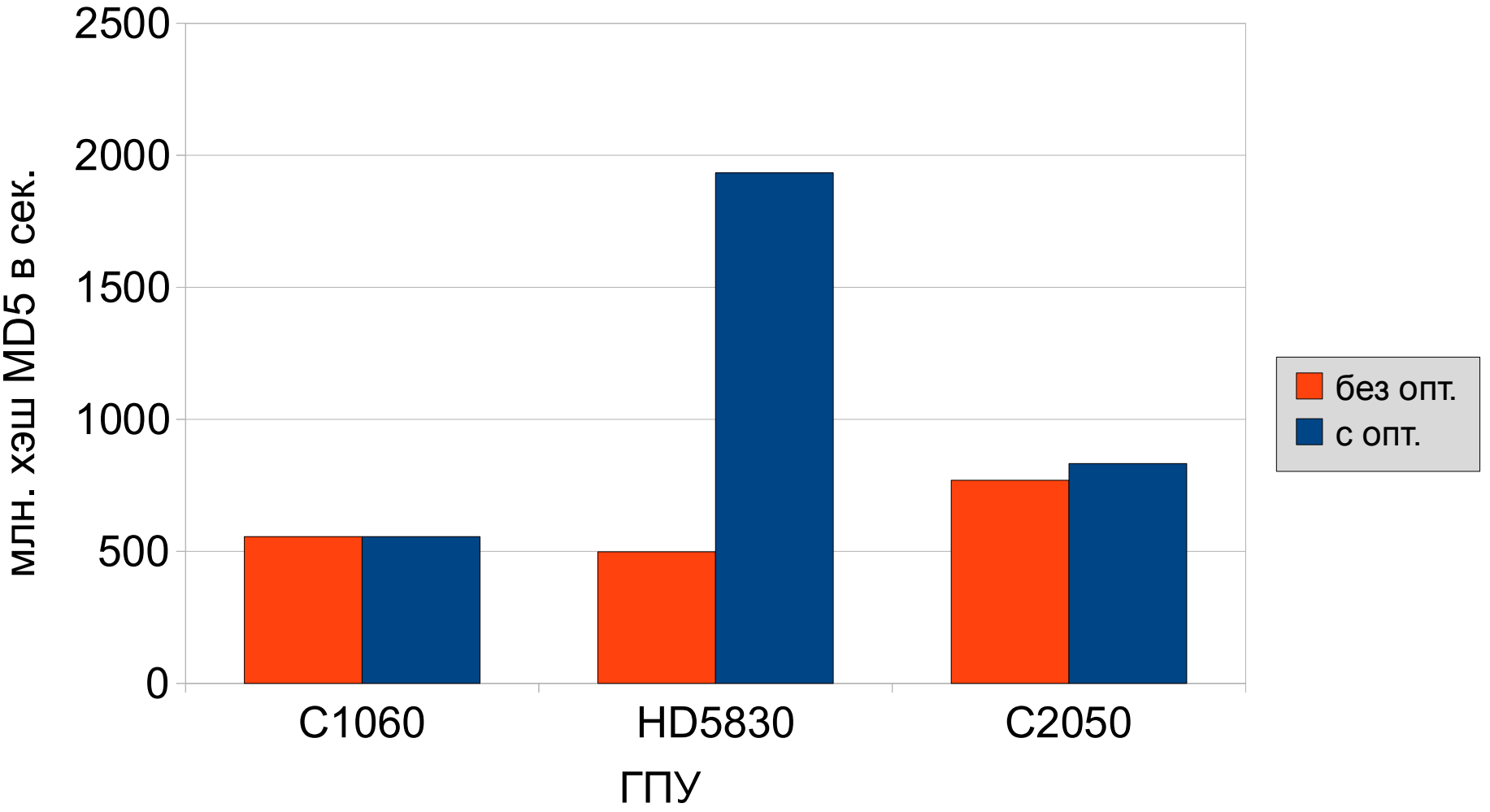
Немного терминологии…

- Метапрограммирование
  - Программы преобразуют другие программы
- Генеративное программирование
  - Порождение новых программ по описанию
- Языко-ориентированное программирование
  - Создание DSL
  - Использование DSL

Хэш-функции: объём кода

- Версия для CUDA
  - 1561 строки кода
- Версия для NUDA
  - 874 строки кода, 4 файла
  - ГПУ NVidia и AMD
  - Поддержка нескольких ГПУ

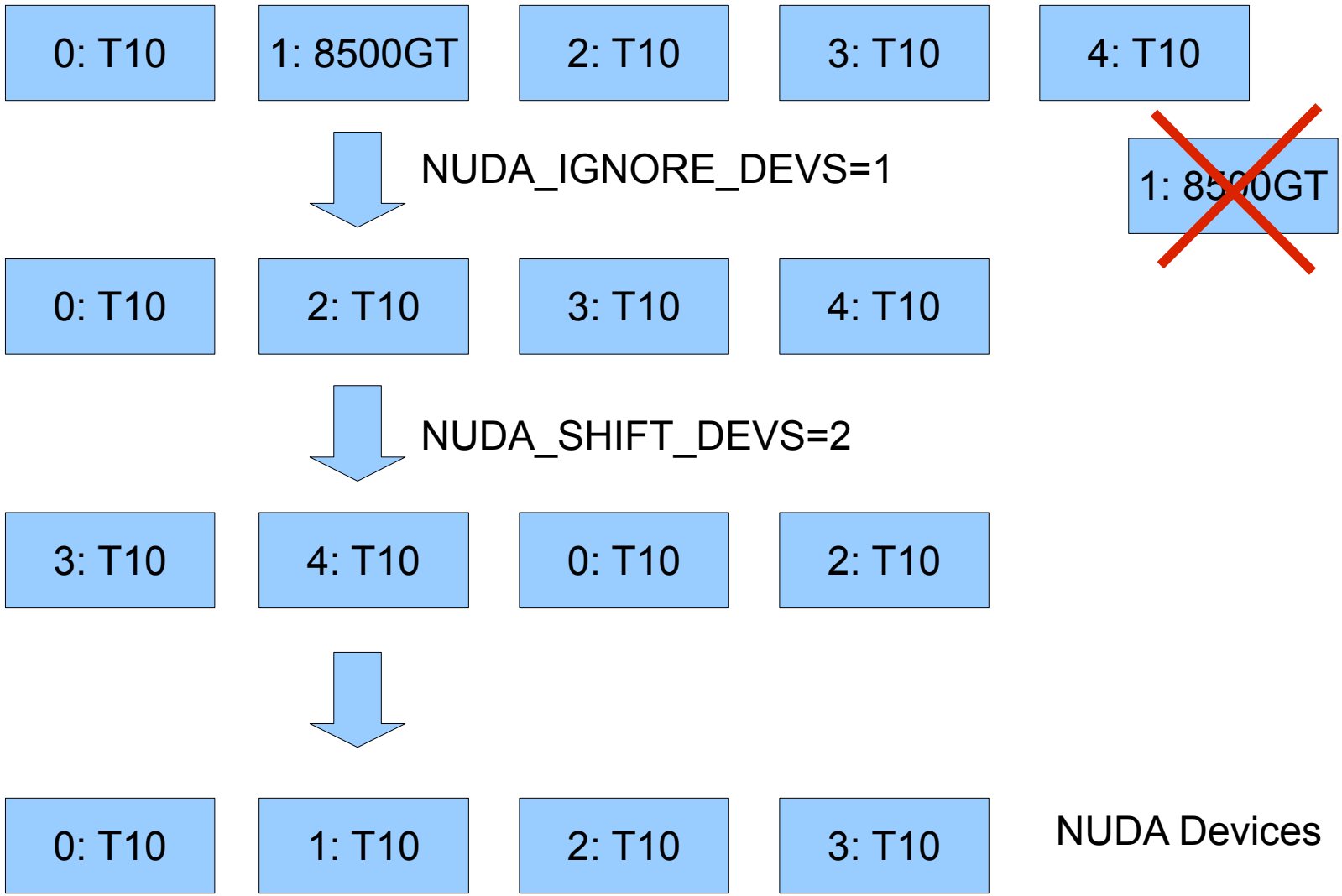
Производительность на MD5



Счётчики производительности NUDA

- **using** Extran.Stat.PerfStat
- Счётчики
  - deviceTime — время исполнения
  - copyTime — время копирования туда-обратно
  - deviceTimeOcl, copyTimeOcl
- Операции
  - get() - значение счётчика
  - reset() - сброс счётчика

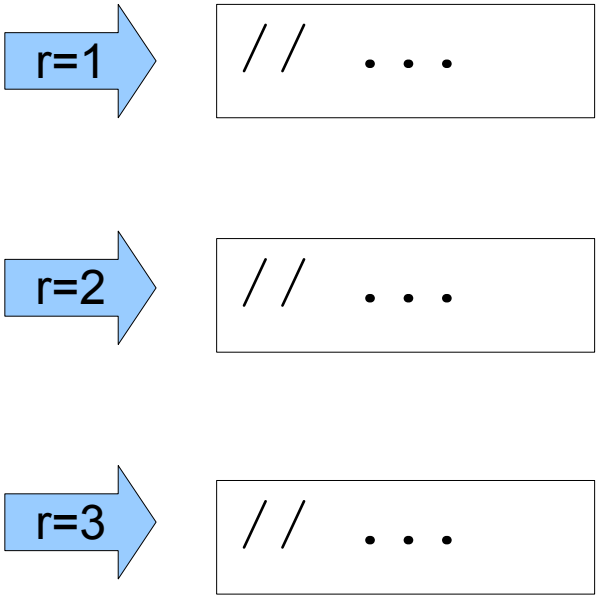
Устройства NUDA



Варианты кода

- компиляция
  - генерируется несколько вариантов кода
- выполнение
  - вариант выбирается по параметру

```
varies(r in (1, 2, 3)) nuwork(1, 128)
nfor((i, j) in (n, n))
  inline nfor((p, q) in
    (-r<>r, -r<>r)){
    res += k[p+1, q+1] *
      a[clamp(i+p, 0, n-1),
      clamp(j+q, 0, n-1)];
  }
}
```



Глобальный барьер на ГПУ

```
nuteam {
  def ls = localSize(0);
  def ltid = localId(0);
  def wgid = groupId(0);
  def ngroups = numGroups(0);
  nfor(k in n) {
    // each thread does its own column(s)
    nfor((i, j) in (wgid <> n - 1 :/ ngroups, ltid <> n - 1 :/ ls)) {
      paths[i, j] = min(paths[i, j], paths[i, k] + paths[k, j]);
    }
    // barrier after each iteration
    globarrier;
  }
}
```