# Методы распределения вычислений при автоматическом распараллеливании непроцедурных спецификаций

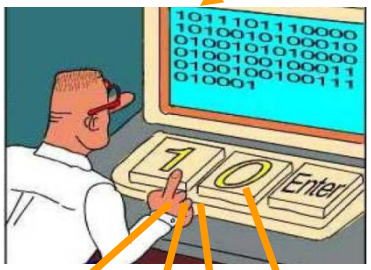
**А.Н. Андрианов, Т.П. Баранова,**
**А.Б. Бугеря, К.Н. Ефимкин**
*Институт прикладной математики им. М.В. Келдыша РАН*

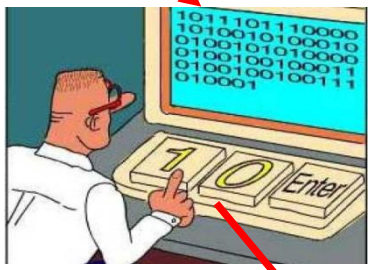
# Процесс разработки программы для решения задачи математической физики

1. Постановка задачи. В качестве исходной информации берется некоторая физическая задача, и строится её математическая модель. Выходом этого этапа является обычно система дифференциальных уравнений с определенными граничными условиями.
2. Выбор пространственно-временной сетки и дискретизация уравнений с помощью одного из разностных методов.
3. Выбор метода решения дискретных уравнений. В результате получаются формулы (соотношения), описывающие необходимые вычисления в точках сетки.
4. Описание, полученное на предыдущем этапе, программируется на некотором языке, который обеспечивает решение задачи на вычислительной машине.

   **Основная идея подхода** – определить язык, позволяющий записать формулы, полученные на этапе 3, в виде непроцедурных спецификаций и создать транслятор-синтезатор, который по спецификациям строит программу для компьютера, то есть исключить или хотя бы свести к минимуму работу человека на этапе 4.

**Формулы, описывающие расчет**




**Программа на CUDA**

**Программа на MPI**

**Программа на OpenMP**

**Программа на ShMem**

**…**

**Программа на языке НОРМА**

**Компилятор**

# Историческая справка

**1963.** Статья И.Б. Задыхайло “Организация циклического процесса счета по параметрической записи специального вида” // Журнал вычислительной математики и математической физики. 1963. Т. 3. N 2.

$$
\begin{aligned}
\overset{1}{X}_i &= f^1(\overset{1}{X}_{i-\Delta_{1,1}}, \overset{2}{X}_{i-\Delta_{1,2}}, \ldots, \overset{n}{X}_{i-\Delta_{1,n}}) \\
\overset{2}{X}_i &= f^2(\overset{1}{X}_{i-\Delta_{2,1}}, \overset{2}{X}_{i-\Delta_{2,2}}, \ldots, \overset{n}{X}_{i-\Delta_{2,n}}) \\
&\ldots \\
\overset{n}{X}_i &= f^n(\overset{1}{X}_{i-\Delta_{n,1}}, \overset{2}{X}_{i-\Delta_{n,2}}, \ldots, \overset{n}{X}_{i-\Delta_{n,n}})
\end{aligned}
$$

Организовать процесс вычисления $\overset{p}{X}_i \quad p = 1, \ldots, n$

для индексов от $m^0(p)$ до $m^N(p), \quad p = 1, \ldots, n, \quad m^0(p)$ заданы.

# Свойства языка НОРМА. Ограничения.

1. Декларативность. Описывается запрос на вычисление, каким образом он реализуется, не указывается. Нет понятий памяти, управления (переходов, циклов и т.п.). Порядок операторов в программе произвольный – компилятор организует порядок вычислений самостоятельно.
2. Однократное присваивание. Single Assignment Language (класс SAL-языков). Нет глобальных переменных и побочных эффектов.
3. Ограничение на вид индексных выражений у величин: $(A*i + C_1)/B + C_2$, где i – индексная переменная, A и B – натуральные константы, а $C_1$ и $C_2$ – целые константы.
4. Области (массивы) имеют границы, заданные константными выражениями, известными в момент трансляции. Области также могут иметь переменные границы, определяемые целочисленными формальными параметрами раздела.

Декларативность --> синтез выходной программы.

Задача синтеза выходной программы в этих ограничениях для языка НОРМА разрешима.

# Язык НОРМА. Пример 1.

Решение системы линейных уравнений:

$$\sum_{j=1}^{m} A_{i,j} X_j = b_i, \quad i = 1,\ldots,m$$

Метод решения

$$X_i^0 = X0, \quad i = 1,\ldots,m$$

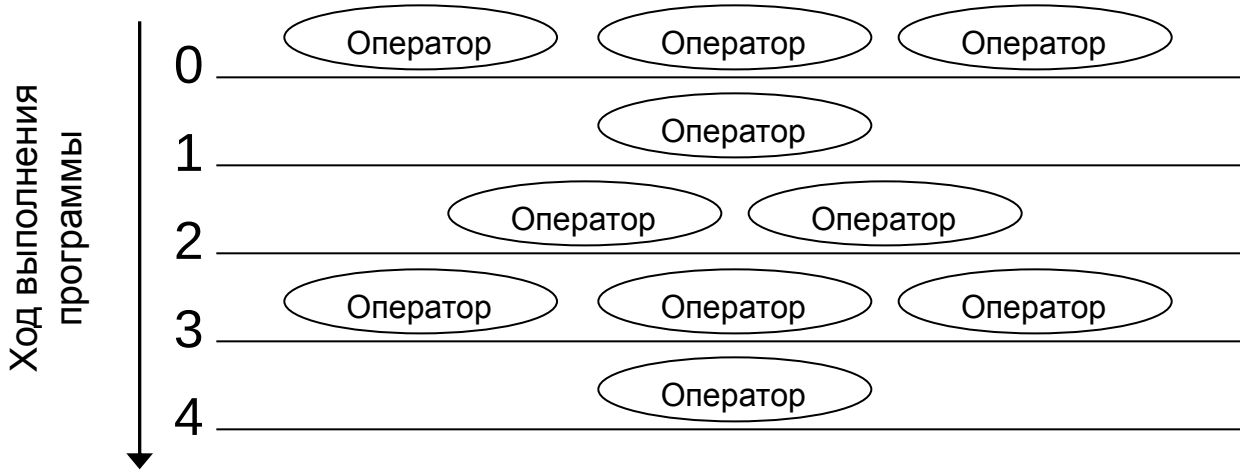
$$X_i^{n+1} = \frac{1}{A_{i,i}}(b_i - \sum_{i \neq j} A_{i,j} X_j^n). \qquad i = 1,\ldots, m$$

Условие выхода $\left\| X_i^n - X_i^{n-1} \right\| < \varepsilon, \; i = 1,\ldots, m$

# Язык НОРМА. Пример 1. Программа.

```
MAIN PART Linear.
BEGIN
Oi: (I=1..M). Oj: (J=1..M). Oij: (Oi; Oj). O1,O2: Oj / J<>I.
VARIABLE X0, X, b DEFINED ON Oi. VARIABLE A DEFINED ON Oij.
VARIABLE E.
DOMAIN PARAMETERS M = 20000. DISTRIBUTION INDEX I = 20.
INPUT X0, b ON Oi. INPUT A ON Oij. INPUT E.
OUTPUT X ON Oi.
ITERATION X ON N.
      INITIAL N = 0:
      FOR Oi ASSUME X = X0.
      END INITIAL
      FOR Oi ASSUME X = 1/A[J=I] * (b - SUM( (O1)A * X[I=J,N-1])).
      EXIT WHEN MAX( (Oi) ABS(X[N] - X[N-1]) ) < E.
END ITERATION N.
END PART.
```

# Трансляция программ на языке НОРМА для параллельных архитектур: ярусно-параллельная форма



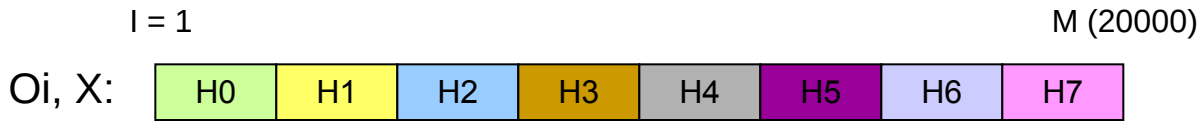

3 уровня параллелизма:

- параллельное выполнение операторов, находящихся на одном ярусе;
- параллельное вычисление значений во всех точках области вычисления оператора (исключение: скалярные операторы, сильно связанные компоненты);
- параллельное вычисление функций редукции.

# Распределение вычислений: OpenMP

**FOR Oi ASSUME X = 1/A[J=I] * (b - SUM((O1) A * X[I=J,N-1])).**

EXIT WHEN **MAX( (Oi) ABS(X[N] - X[N-1] ) )** < E**.**

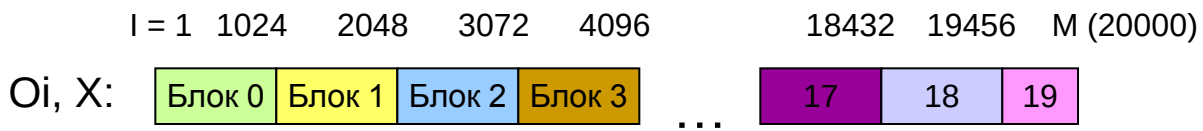



```
#pragma omp for private(J, Total)
for(I = 0; I <= M-1; I++) {
  Total = 0.0;
  for(J = 0; J <= M-1; J++) {
    if(J+1 != I+1)
      Total = Total + A[I][J] * XShadow[J];
  }
  X[I] = 1 / A[I][I] * (b[I] - Total);
}
```

```
#pragma omp parallel private(Priv)
#pragma omp single
{
  Total1 = abs(X[1-1] - (XShadow[1-1]));
}
Priv = abs(X[1-1] - (XShadow[1-1]));
#pragma omp for private(Tmp)
for(I = 1; I <= M; I++) {
  Tmp = abs(X[I-1] - (XShadow[I-1]));
  if(Priv < Tmp)
    Priv = Tmp;
}
#pragma omp critical
{
  if(Total1 < Priv)
    Total1 = Priv;
}
```

# Распределение вычислений: NVIDIA CUDA

**FOR Oi ASSUME X = X0.**



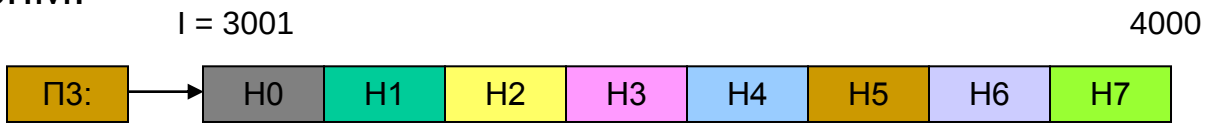

```
static __global__ void kernel0(void)
{
  int I = threadIdx.x + blockIdx.x*1024;
  if(I < M)
    XShadow_dev[I] = X0_dev[I];
}

int main(int argc, char **argv)
{
  ……
  kernel0<<< 20, 1024 >>>();
 ……
}
```

# Распределение вычислений: MPI+…

**DISTRIBUTION INDEX I = 20.**
**FOR Oi ASSUME X = X0.**

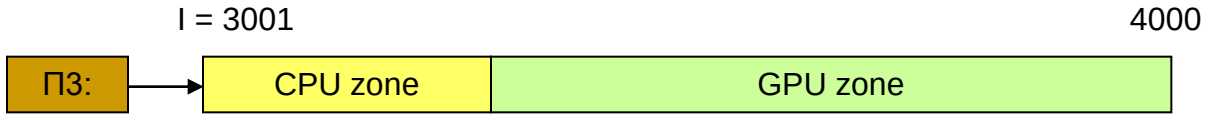

```
for(I = 0; I <= 999; I++) {
……….
```

+ OpenMP




```
#pragma omp for
for(I = 0; I <= 999; I++) {
……….
```

+ OpenMP+CUDA



# Результаты применения компилятора: прикладная задача из области газодинамики

В основном рабочем итерационном цикле данной программы производится расчёт различных величин для всех точек одномерной области. Для каждой точки вызывается «плоский» внешний раздел, который на основе предыдущих значений точки и её соседей высчитывает данные для новых значений. Все вычисления производятся с одинарной точностью.

| Последовательная программа, 1 ядро Intel Xeon X5670 | OpenMP программа, 12 ядер Intel Xeon X5670 | CUDA программа, 1 nVidia Fermi C2050 | MPI программа, 48 MPI процессов, 4 узла по 12 ядер Intel Xeon X5670 | MPI+OpenMP программа, 4 MPI процесса, 4 узла по 12 ядер Intel Xeon X5670 |
|---|---|---|---|---|
| 461 сек | 45.6 сек | 16.9 сек | 15.8 сек | 22.1 сек |

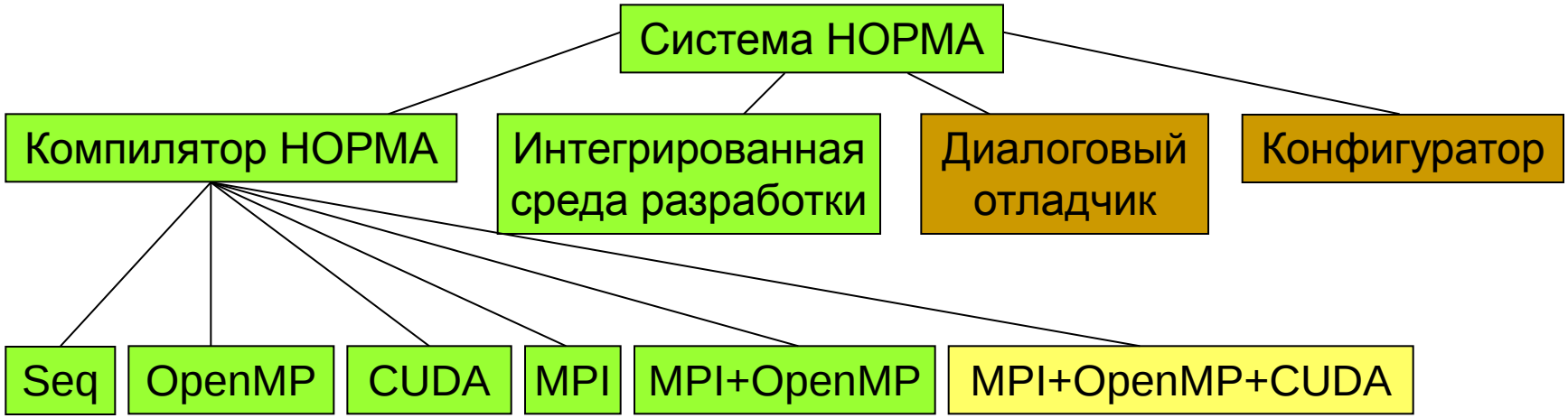
# Результаты применения компилятора: реализация теста CG из пакета NPB

Тестовое ядро CG (Conjugate Gradient) осуществляет приближение к наименьшему собственному значению большой разреженной симметричной положительно определенной матрицы с использованием метода обратной итерации вместе с методом сопряженных градиентов в качестве подпрограммы для решения СЛАУ.

«Сердце» теста – операция умножения разреженной матрицы на вектор. 3 варианта реализации: простая (8 строк), редукция (120 строк), cuSPARSE (60 строк)

| | Class A (Mop/s) | | Class B (Mop/s) | | Class C (Mop/s) | |
|---|---|---|---|---|---|---|
| | Си | Си + НОРМА | Си | Си + НОРМА | Си | Си + НОРМА |
| Последовательная программа, 1 ядро Intel Xeon X5670 | 1200 | 1175 | 718 | 716 | 513 | 615 |
| OpenMP программа, 12 ядер Intel Xeon X5670 | 9737 | 8035 | 3595 | 3004 | 3240 | 2806 |
| CUDA программа, простая реализация, 1 nVidia Fermi C2050 | — | 927 | — | 722 | — | 645 |
| CUDA программа, реализация с редукцией, 1 nVidia Fermi C2050 | — | 2134 | — | 2178 | — | 1876 |
| CUDA программа, использование cuSPARSE, 1 nVidia Fermi C2050 | — | 6521 | — | 4504 | — | 2704 |

# Система НОРМА



# Заключение

Применение языка НОРМА для решения прикладных задач из области математической физики дает:

1. Высокий уровень абстракции при составлении программы прикладным специалистом, что снижает количество возможных ошибок в программе.
2. Позволяет получать эффективные исполняемые программы для различных видов параллельных архитектур.
3. При этом сама прикладная программа остаётся неизменной (за исключением, возможно, внешних функций на целевом языке программирования).
4. Компилятор автоматически решает все задачи, специфичные для заданной архитектуры, в том числе по распараллеливанию программы, распределению вычислений, балансировке вычислительной нагрузки и обмену данными между различными вычислительными устройствами.

# Спасибо за внимание!

http://www.keldysh.ru/pages/norma/

shurabug@yandex.ru

Работа выполнена при
финансовой поддержке гранта
РФФИ № 18-01-00131-а.