О подходах к преобразованию программ в системе САПФОР *

М.С. Баранов[12], Н.А. Катаев[1], А.А. Смирнов[1]
Институт Прикладной Математики им. М.В. Келдыша РАН [1], Московский Государственный Университет им. М.В. Ломоносова[2]

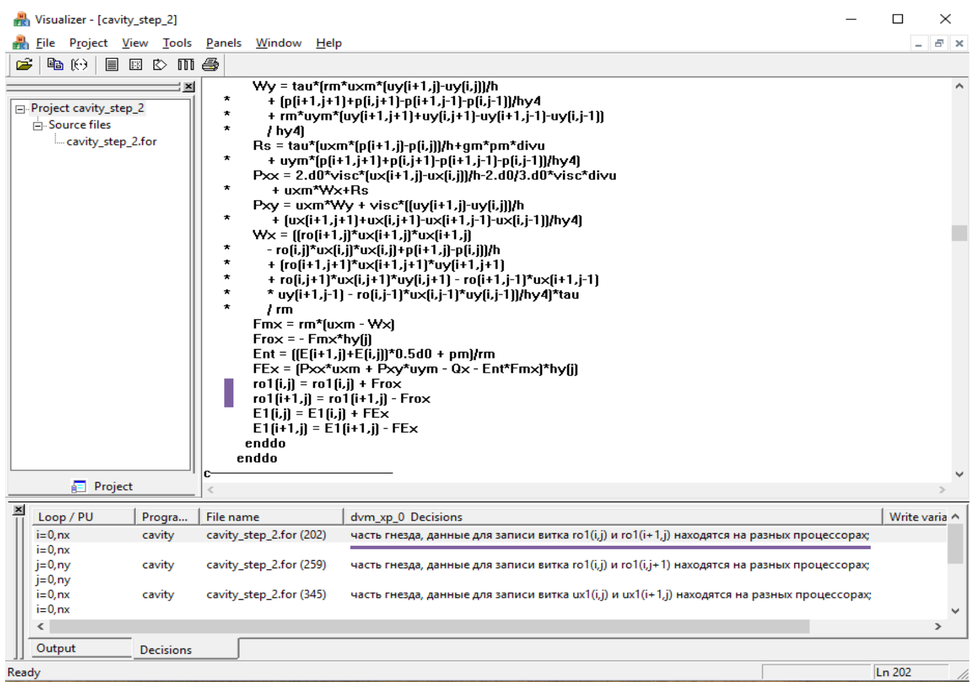
Параллельные вычислительные технологии (ПаВТ'2016)
Parallel Computational Technologies (PCT'2016)

* Работа поддержана грантом РФФИ 14-07-31321

Результат применения фиксированной, программно-независимой последовательности оптимизаций при компиляции программ не отражает особенности их информационной структуры. Это сказывается на эффективности параллельных программ, получаемых при применении неадаптивных автоматически распараллеливающих компиляторов. Итерационный процесс распараллеливания программ, поддерживаемый системой автоматизированного распараллеливания САПФОР, позволяет выбирать нужные преобразования и применять их только, когда они необходимы для устранения проблем, препятствующих распараллеливанию.

Предлагается два подхода к поиску оптимизационных последовательностей, описывающих преобразования.

Первый подход состоит в автоматическом выборе способа устранения проблемы. В системе САПФОР предусмотрен набор оптимизационных последовательностей, которые применяются, если указанная проблема удовлетворяет определенным критериям.




```
do j = 1,ny
 do i = 0,nx
  hy4 = hy(j+1)+2.d0*hy(j)+hy(j-1)
  ...
  Frox = ...
  ro1(i,j) = ro1(i,j) + Frox
  ro1(i+1,j) = ro1(i+1,j) - Frox
  E1(i,j) = E1(i,j) + FEx
  E1(i+1,j) = E1(i+1,j) - FEx
 enddo
enddo
```

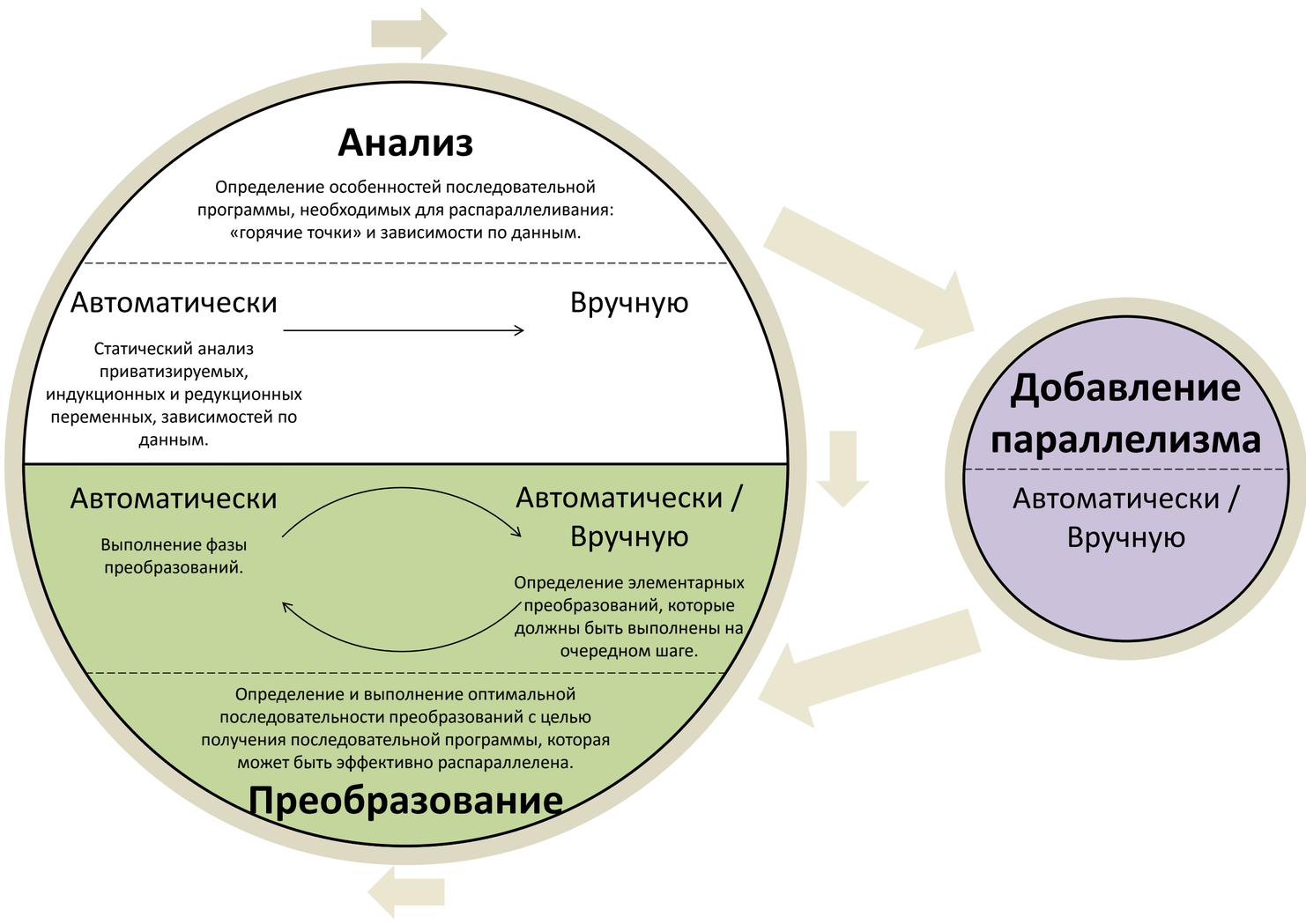
```
do j = 1,ny
 do i = 0,nx
  hy4 = hy(j+1)+2.d0*hy(j)+hy(j-1)
  ...
  Frox = ...
  tmp1(i+1,j) = Frox
  ro1(i+1,j) = ro1(i+1,j) – Frox
  E1(i,j) = E1(i,j) + FEx
  E1(i+1,j) = E1(i+1,j) - FEx
 enddo
enddo

do i = 1,nx
 do j = 0, ny
  ro1(i,j) = ro1(i,j) + tmp1(i+1,j)
 enddo
enddo
```

«данные для записи витка ro1(i, j) и ro(i,j+1) находятся на разных процессорах»

Альтернативой данному подходу является полуавтоматическое преобразование программ с активным участием пользователя системы. Система предоставляет пользователю набор элементарных преобразований, которые могут быть выполнены автоматически. Используя данные преобразования в качестве строительных блоков, пользователь составляет из них оптимизационные последовательности, указывая участки кода, которые должны быть преобразованы, и задавая параметры преобразований.

Автоматизация распараллеливания



Автоматизированное распараллеливание программ Спекание 2D и Спекание 3D

При активном участии пользователя были распараллелены прикладные программы, реализующих двухмерную и трехмерную задачи моделирования процессов плавления многокомпонентных порошков при селективном лазерном спекании на основе многокомпонентной и многофазной гидродинамической модели.

Распараллеливание выполнялось с использованием технологий параллельного программирования OpenMP, OpenACC, DVMH.

Преобразования	Последовательность элементарных преобразований для программы Спекание 2D									Итого
	1	2	3	4	5	6	7	8	9	
Подстановка переменных	8									8
Вынесение инвариантов	1									1
Разворачивание циклов	5	1								10
Распределение циклов			1	2			4	2		5
Перестановка циклов					1				2	3
Объединение циклов					4				2	6
Перестановка циклов										0
Сдвиг индекса						2				2
Итого	14	1	1	2	5	2	4	2	4	35

```
for (I = 1; I <= Nx; I += 1) {/*access to Flx[I][0]*/}
for (I = 1; I <= Nx; I += 1)
  for (J = 1; J <= Nz - 1; J += 1) {/*access to Flx[I][J]*/}
for (I = 1; I <= Nx; I += 1) {/*access to Flx[I][Nz]*/}
for (J = 1; J <= Nz; J += 1) {/*access to Flz[0][J]*/}
for (I = 1; I <= Nx - 1; I += 1)
  for (J = 1; J <= Nz; J += 1) {/*access to Flz[I][J]*/}
for (J = 1; J <= Nz; J += 1) {/*access to Flz[Nx][J]*/}
```

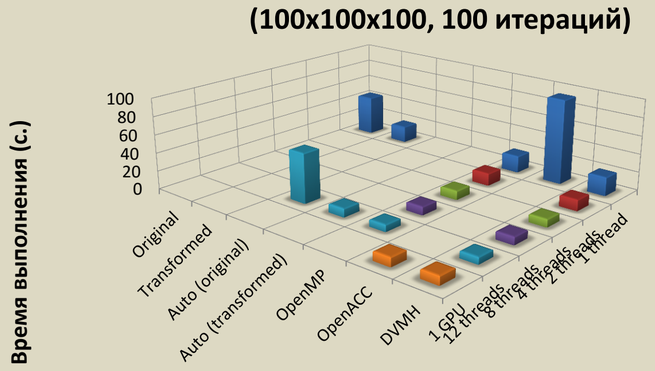
⇦ ••• ⇦

```
for (I = 1; I < Nx + 1; ++I)
  for (J = 1; J < Nz + 1; ++J)
    #pragma x unroll
    for (II = 0; II < 2; ++II)
      for (JJ = 0; JJ < 2; ++JJ) {
        ...
        Flx[I][J - 1 + JJ] = Flx[I][J - 1 + JJ] - Iloc[0] * Fl[0];
        Flz[I - 1 + II][J] = Flz[I - 1 + II][J] - Jloc[1] * Fl[1];
      }
```

⇩

```
#pragma x split(Flx, Flz)
for (I = 1; I < Nx + 1; ++I)
  for (J = 1; J < Nz + 1; ++J) {
    ...
    Flx[I][J - 1] = Flx[I][J - 1] - 1 * Fl[0];
    Flz[I - 1][J] = Flz[I - 1][J] - 1 * Fl[1];
    ...
    Flx[I][J] = Flx[I][J] - 1 * Fl[0];
    Flz[I - 1][J] = Flz[I - 1][J] - -1 * Fl[1];
    ...
    Flx[I][J - 1] = Flx[I][J - 1] - -1 * Fl[0];
    Flz[I][J] = Flz[I][J] - 1 * Fl[1];
    ...
    Flx[I][J] = Flx[I][J] - -1 * Fl[0];
    Flz[I][J] = Flz[I][J] - -1 * Fl[1];
}
```

Время выполнения (секунды) программы Спекание 3D (100x100x100, 100 итераций)

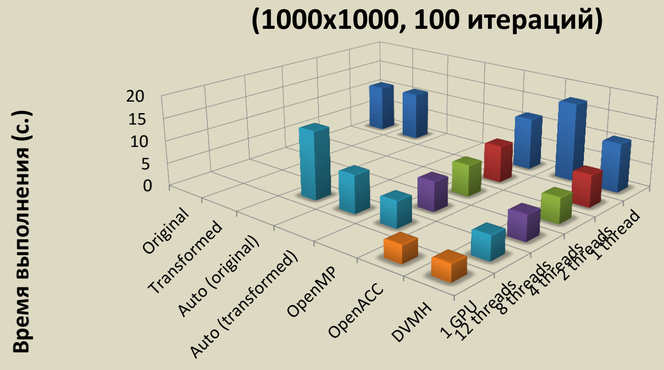


	Original	Transformed	Auto (original)	Auto (transformed)	OpenMP	OpenACC	DVMH
1 thread	44,46	18,64			18,7	93,74	20,7
2 threads					13,55		12,29
4 threads					9,82		8,68
8 threads					9,16		8,63
12 threads			54,64	9,41	7,61		7,46
1 GPU						9,74	9,38

Вычислительная система

- CPU: 6-ядерный процессор Intel(R) Xeon(R) CPU E5-1660 v2 3.70GHz с включенным Hyper Threading (2 нити на ядро);
- GPU: NVIDIA GTX Titan (поколение Kepler).

Время выполнения (секунды) программы Спекание 2D (1000x1000, 100 итераций)



	Original	Transformed	Auto (original)	Auto (transformed)	OpenMP	OpenACC	DVMH
1 thread	10,73	11,02			11,57	17,23	10,79
2 threads					8,11		7,12
4 threads					6,9		5,59
8 threads					6,52		5,58
12 threads			15,04	8,35	5,79		5,13
1 GPU						3,75	3,6

Компиляторы

- Intel C Compiler V16.0.2 с опцией -O3 для последовательных, OpenMP- и DVMH-программ. Также приведены результаты автоматического распараллеливания с опцией –parallel.
- NVIDIA CUDA Toolkit V7.5.17 с опцией –O3 для DVMH-программ.
- PGI V15.7-0 с опцией -O3 для OpenACC-программ.